O'REILLY®

Машинное обучение и безопасность

Кларенс Чио
Дэвид Фримэн

ДМК ПРЕСС
ИЗДАТЕЛЬСТВО

Clarence Chio and David Freeman

Machine Learning and Security

Protecting Systems with Data and Algorithms

Beijing • Boston • Farnham • Sebastopol • Tokyo

O'REILLY®

Кларенс Чио, Дэвид Фримэн

Машинное обучение и безопасность

Защита систем с помощью данных и алгоритмов

ДМК ПРЕСС
издательство
Москва, 2020

УДК 004.89, 004.492
ББК 32.972
Ч58

Чио К., Фримэн Д.
Ч58 Машинное обучение и безопасность / пер. с анг. А. В. Снастина. – М.: ДМК Пресс, 2020. – 388 с.: ил.

ISBN 978-5-97060-713-8

Способна ли технология машинного обучения решить проблемы компьютерной безопасности? Или надежда на это является лишь следствием повышенного внимания к машинному обучению?

С помощью этой книги вы изучите способы применения машинного обучения в задачах обеспечения безопасности, таких как выявление вторжения извне, классификация вредоносных программ и анализ сетевой среды. Особое внимание уделено задачам по созданию работоспособных, надежных масштабируемых систем извлечения и анализа данных в сфере обеспечения безопасности.

Издание предназначено инженерам по обеспечению безопасности, а также специалистам по обработке данных научными методами.

УДК 004.89, 004.492
ББК 32.972

ISBN 978-1-491-97990-7 (анг.)
ISBN 978-5-97060-713-8 (рус.)

Отзывы

Будущие меры по обеспечению безопасности и защиты должны определяться способностью защищающихся развертывать средства машинного обучения для быстрого обнаружения и прекращения деятельности злоумышленников в интернете в любых масштабах. Чио и Фримэн написали по этой теме исчерпывающе полную книгу, включающую самые последние достижения научной мысли, а также трудные для изучения практические методики развертывания средств машинного обучения с целью обеспечения защиты людей в этой сфере деятельности.

– Алекс Стамос,
руководитель службы безопасности Facebook

Превосходное практическое руководство для всех, кто намерен освоить использование технологии машинного обучения для обеспечения безопасности компьютерных систем, от выявления аномалий до защиты конечных пользователей.

– Дэн Боне,
профессор ИТ, Стэнфордский университет

Если вы хотите знать, какое место занимает машинное обучение в области обеспечения безопасности, то книга Чио и Фримэна даст вам четкое представление об этом.

– Нвокеди С. Идика,
доктор наук, инженер ПО, Google, Security & Privacy Organization

Содержание

Предисловие

Машинное обучение завоевывает мир. Коммуникации и связь, финансовая сфера, транспорт, производство товаров и даже сельское хозяйство[1] – практически каждая отрасль технологии изменилась под влиянием машинного обучения или изменится в ближайшем будущем.

Обеспечение компьютерной безопасности также является важнейшей проблемой для всего мира. Поскольку мы становимся все более зависимыми от компьютеров в работе, развлечениях и вообще в обычной жизни, в равных пропорциях возрастает и значимость наличия брешей и лазеек в компьютерных системах, привлекающих нездоровое внимание постоянно увеличивающегося круга атакующих злоумышленников, которые надеются такими способами получить деньги или просто причинить ущерб. Более того, поскольку системы становятся все более сложными и взаимосвязанными, все труднее обеспечить отсутствие в них ошибок и непредвиденных лазеек, которые открывают доступ атакующим. Уже после сдачи книги в печать мы узнали о том, что в настоящее время слишком много микропроцессоров (если не каждый) используется без надлежащей защиты[2].

Машинное обучение предлагает (потенциальные) решения в любой области деятельности, поэтому вполне естественно, что эта технология применима и для компьютерной безопасности, т. е. для области, которая по своей сути является источником полезных и надежных наборов данных, на основе которых, собственно, и развивается технология машинного обучения. В самом деле, во всех сообщениях об угрозах для безопасности, которые появляются в новостях, обнаруживается аналогичное количество заявлений о том, что искусственный интеллект может «совершить революцию» в области методик обеспечения безопасности. Надежды на полное уничтожение наиболее значимых преимуществ атакующих злоумышленников привели к тому, что машинное обучение было широко разрекламировано как технология, которая позволит наконец завершить длительную игру в кошки-мышки между атакующими и защищающимися. При посещении экспозиционных залов самых крупных конференций по безопасности обнаруживается следующая тенденция: все большее количество компаний начинает использовать машинное обучение для решения проблем безопасности.

Быстро растущая заинтересованность в объединении этих двух областей порождает еще и атмосферу цинизма, в которой отвергается сама идея объединения – считается, что вокруг нее создан нездоровый ажиотаж. Как найти разумный баланс? Каков реальный потенциал применения методов искусственного интеллекта в сфере обеспечения безопасности? Как отличить рекламную шумиху от действительно многообещающих технологий? Что должен взять на вооружение конкретный пользователь для решения своих проблем безопасности? Мы ре-

---

[1] Monsanto. How Machine Learning is Changing Modern Agriculture. Modern Agriculture. September 13, 2017. **https://modernag.org/innovation/machine-learning-changing-modern-agriculture/**.

[2] Meltdown and Spectre. Graz University of Technology, accessed January 23, 2018. **https://spectreattack.com/**.

шили, что наилучшим вариантом ответов на все эти вопросы является глубокое изучение научных основ и методик, понимание главных концепций, огромный объем практического тестирования и экспериментирования, чтобы полученные результаты говорили сами за себя. Но все это требует большого объема актуальных знаний как в области датологии (науки о данных), так и в области обеспечения компьютерной безопасности. В процессе нашей работы по созданию систем безопасности, руководства группами, противодействующими злоупотреблениям, а также при участии в конференциях мы встретили некоторых людей, которые обладают таким объемом знаний. Кроме того, многие хорошо знают одну из этих областей и намерены изучать другую.

В результате появилась на свет данная книга.

О чем эта книга

Эта книга была написана, чтобы предоставить рабочую платформу для обсуждения неизбежного объединения двух широко распространенных концепций: машинного обучения и безопасности. Существует некоторое количество литературы, объединяющей эти две темы (а также многочисленные рабочие группы и семинары конференций: CCS AISec (**http://ai-sec.net**), AAAI AICS (**http://www-personal.umich.edu/~arunesh/AICS2018/**), NIPS Machine Deception (**https://www.machinedeception.com/**)), но большинство опубликованных работ является научными или теоретическими. Нам не удалось найти ни одного руководства, в котором были бы представлены конкретные работающие примеры с исходным кодом, способные помочь специалистам по обеспечению безопасности освоить науку о данных, а специалистам по машинному обучению в полной мере овладеть современными методиками решения задач обеспечения безопасности.

Исследуя широкий спектр тем в области обеспечения безопасности, мы представили примеры возможного практического применения технологии машинного обучения для улучшения или замены основанных на правилах или эвристических решениях таких задач, как обнаружение вторжения, классификация вредоносных программ и анализ сетевой среды. В дополнение к изучению основных алгоритмов и методик машинного обучения особое внимание мы уделили трудным задачам по созданию работоспособных, надежных, масштабируемых систем извлечения и анализа данных в сфере обеспечения безопасности. С помощью реально работающих примеров и подробных обсуждений с разъяснениями мы демонстрируем, как воспринимать и интерпретировать данные в конкурирующей среде и как обнаружить и выделить важные сигналы, которые могут быть незаметными в общем «шумовом» фоне.

Для кого предназначена эта книга

Если вы работаете в сфере обеспечения безопасности и намерены использовать машинное обучение для усовершенствования контролируемых вами систем, то эта книга для вас. Если вы специалист по машинному обучению и намерены использовать эту технологию для решения задач по обеспечению безопасности, то эта книга будет полезной и для вас.

Предполагается, что читатель обладает основами знаний о математической статистике, но большинство более сложных математических выкладок при первом чтении можно пропустить без ущерба для понимания концепций и принципов. Кроме того, предполагается знание какого-либо языка программирования. Примеры в книге написаны на языке программирования Python, также приводятся ссылки на пакеты Python, необходимые для реализации рассматриваемых концепций, но вы можете самостоятельно реализовать эти концепции, используя библиотеки с открытым исходным кодом на языках Java, Scala, C++, Ruby и многих других языках программирования.

Условные обозначения и соглашения, принятые в книге

В книге используются следующие типографские соглашения.

Курсив используется для смыслового выделения важных положений, новых терминов, URL-адресов и адресов электронной почты в интернете, имен команд и утилит, а также имен и расширений файлов и каталогов.

`Моноширинный шрифт` используется для листингов программ, а также в обычном тексте для обозначения имен переменных, функций, типов, объектов, баз данных, переменных среды, операторов, ключевых слов и других программных конструкций и элементов исходного кода. Также применяется для команд, выполняемых в командной строке, и для вывода результатов их выполнения.

`Моноширинный полужирный шрифт` используется для обозначения команд или фрагментов текста, которые пользователь должен ввести дословно без изменений. Также применяется для выделения важных фрагментов в выводимых результатах.

`Моноширинный курсив` используется для обозначения в исходном коде или в командах шаблонных меток-заполнителей, которые должны быть заменены соответствующими контексту реальными значениями.

Эта пиктограмма обозначает совет, рекомендацию или примечание общего характера.

Эта пиктограмма обозначает предупреждение или особое внимание к потенциально опасным объектам.

Отзывы и пожелания

Мы всегда рады отзывам наших читателей. Расскажите нам, что вы думаете об этой книге – что понравилось или, может быть, не понравилось. Отзывы важны для нас, чтобы выпускать книги, которые будут для вас максимально полезны.

Вы можете написать отзыв прямо на нашем сайте **www.dmkpress.com**, зайдя на страницу книги, и оставить комментарий в разделе «Отзывы и рецензии». Также можно послать письмо главному редактору по адресу **dmkpress@gmail.com**, при этом напишите название книги в теме письма.

Если есть тема, в которой вы квалифицированы, и вы заинтересованы в написании новой книги, заполните форму на нашем сайте по адресу **http://dmkpress.com/authors/publish_book/** или напишите в издательство по адресу **dmkpress@gmail.com**.

Скачивание исходного кода примеров

Скачать файлы с дополнительной информацией для книг издательства «ДМК Пресс» можно на сайте **www.dmkpress.com** или **www.дмк.рф** на странице с описанием соответствующей книги.

Список опечаток

Хотя мы приняли все возможные меры для того, чтобы удостовериться в качестве наших текстов, ошибки все равно случаются. Если вы найдете ошибку в одной из наших книг — возможно, ошибку в тексте или в коде, — мы будем очень благодарны, если вы сообщите нам о ней. Сделав это, вы избавите других читателей от расстройств и поможете нам улучшить последующие версии этой книги.

Если вы найдете какие-либо ошибки в коде, пожалуйста, сообщите о них главному редактору по адресу **dmkpress@gmail.com**, и мы исправим это в следующих тиражах.

Нарушение авторских прав

Пиратство в интернете по-прежнему остается насущной проблемой. Издательства «ДМК Пресс» и O'Reilly очень серьезно относятся к вопросам защиты авторских прав и лицензирования. Если вы столкнетесь в интернете с незаконно выполненной копией любой нашей книги, пожалуйста, сообщите нам адрес копии или веб-сайта, чтобы мы могли применить санкции.

Пожалуйста, свяжитесь с нами по адресу электронной почты **dmkpress@gmail.com** со ссылкой на подозрительные материалы.

Мы высоко ценим любую помощь по защите наших авторов, помогающую нам предоставлять вам качественные материалы.

Благодарности

Авторы благодарят Хайрама Андерсона (Hyrum Anderson), Джейсона Крэйга (Jason Craig), Нвокеди Идика (Nwokedi Idika), Джесса Мэйлза (Jess Males), Энди Орэма (Andy Oram), Алекса Пинто (Alex Pinto) и Джошуа Сакса (Joshua Saxe) за подробные технические рецензии и отзывы на предварительные черновые версии этой книги. Мы также благодарим Вирджинию Уилсон (Virginia Wilson), Кристен Браун (Kristen Brown) и весь коллектив издательства O'Reilly, оказавший помощь в процессе превращения нашего проекта из концепции в реальность.

Кларенс благодарен Кристине Чжоу (Christina Zhou) за понимание и терпение в течение всех бесконечных ночей и выходных, которые были посвящены написанию этой книги, Ик Лун Ли (Yik Lun Lee) за корректуру черновых версий и обнаружение ошибок в коде, Джерроду Оверсону (Jarrod Overson), который заставил поверить в то, что я смогу сделать это, а также чихуа-хуа Дэйзи, которая всегда была на моей стороне в самые трудные времена. Спасибо Энто Джозефу (Anto Joseph) за обучение науке обеспечения безопасности и всем прочим хакерам, исследователям и участникам учебных курсов, которые так или иначе повлияли на создание этой книги, моим коллегам в Shape Security, которые сделали меня более опытным инженером, а также докладчикам и участникам конференции Data Mining for Cyber Security за то, что они являются частью сообщества, выполняющего исследования в этой области. Но самая большая благодарность – моей семье в Сингапуре за поддержку вне зависимости от того, где я нахожусь, за то, что позволили осуществиться моим мечтам и поощряли мои стремления.

Дэвид благодарен Дипаку Агарвалу (Deepak Agarwal), который убедил его заняться написанием этой книги, Дэну Боне (Dan Boneh) за обучение образу мышления, свойственному специалисту в сфере обеспечения безопасности, а также Винсенту Сильвиера (Vicente Silveira) и коллегам по LinkedIn и Facebook за то, что показали мне, чем в действительности является безопасность в реальном мире. Спасибо Грейс Тан (Grace Tang) за отзыв о разделах, посвященных машинному обучению, и за «случайного пингвина» (occasional penguin). И самая большая благодарность – Торри (Torrey), Илоди (Elodie) и Фебу (Phoebe), которые смирились с моим отсутствием многими поздними вечерами и несколько необычным поведением из-за необходимости завершения работы над этой книгой. У меня никогда не было повода для сомнений в их поддержке.

Глава 1

Машинное обучение и безопасность

В начале был спам.

Как только ученые соединили достаточное количество компьютеров через интернет для создания сети обмена информацией, приносящей реальную пользу, другие люди обнаружили, что это средство свободной передачи и широкого распространения данных представляет собой превосходный способ рекламирования сделанной наспех продукции, похищения секретных учетных и регистрационных данных и распространения компьютерных вирусов.

За следующие 40 лет в отрасль компьютерной и сетевой безопасности было включено значительное количество подобластей и средств противостояния потенциальным угрозам: обнаружение вторжения, защита веб-приложений, анализ вредоносных программ, защита социальных сетей, противодействие постоянно существующим опасностям, практическое применение криптографии и многое другое. Но даже в наши дни проблема спама остается главной для всех, кто имеет дело с электронной почтой или с системой обмена сообщениями. Возможно, вездесущий спам стал едва ли не основной проблемой в сфере обеспечения компьютерной безопасности, проблемой, которая напрямую воздействует на нашу жизнь.

Машинное обучение было изобретено не борцами со спамом, но технические специалисты, сведущие в математической статистике, быстро адаптировали эту технологию, потому что увидели в ней потенциальное средство борьбы с нарастающим потоком злоупотреблений. Провайдеры электронной почты и прочих сервисов интернета (ISP – Internet Service Provider) получили доступ к огромному объему содержимого e-mail-сообщений, метаданных, а также возможность наблюдать за поведением пользователя. С использованием данных сообщений электронной почты можно сформировать основанные на содержимом (контенте) модели для создания обобщенных методик распознавания спама. Метаданные и характерные репутационные объекты можно извлекать из e-mail-сообщений, для того чтобы предварительно определить вероятность того, что электронное письмо является спамом даже без обращения к его содержимому. После создания цикла обратной связи с поведением пользователя вся система в целом может сформировать «коллективный разум», который со временем будет совершенствоваться с помощью самих пользователей.

Таким образом, фильтры электронной почты постепенно развивались, чтобы успешно противодействовать разнообразным хитроумным способам, которые

непрерывно изобретают авторы спама. Даже если 85 % всех сообщений электронной почты, пересылаемых в наши дни, являются спамом (если верить данным одной из исследовательских групп[1]), самые эффективные современные фильтры блокируют более 99.9 % всего спама[2]. Поэтому пользователи наиболее известных сервисов электронной почты чрезвычайно редко обнаруживают неотфильтрованный и нераспознанный спам в своих почтовых ящиках для входящих писем. Это говорит о значительном превосходстве современных методик над упрощенными методиками фильтрации спама, разработанными на начальном этапе использования интернета и использующими простую фильтрацию слов и репутационные характеристики метаданных e-mail-сообщений (**http://www.paulgraham.com/spam.html**) для достижения весьма скромных результатов.

Основным уроком, который извлекли и исследователи-теоретики, и инженеры-практики из этого противостояния, является важность использования данных, для того чтобы одержать победу над противниками-злоумышленниками, а также улучшение качества взаимодействия пользователей (т. е. фактически всех нас) с технологическими достижениями. В действительности история борьбы со спамом представляет собой характерный пример использования данных и технологии машинного обучения в любой области обеспечения компьютерной безопасности. В наши дни почти все организации в значительной мере полагаются на технологические достижения, но почти каждый элемент любой технологии имеет свои уязвимые места. Руководствуясь теми же основными побуждениями, что и спамеры из 1980-х (нерегулируемый свободный и бесплатный доступ к аудитории, располагающей некоторыми средствами и частной закрытой информацией), злоумышленники способны сделать потенциально опасными почти все аспекты нашей современной жизни. Действительно, сама сущность противостояния атакующей и защищающейся сторон одинакова во всех областях компьютерной безопасности, как и в борьбе со спамом: имеющий конкретный стимул злоумышленник постоянно пытается некорректно использовать компьютерную систему. При этом стороны все время пытаются залатать или воспользоваться «дырами» в проектном решении или в технологии, прежде чем другая сторона обнаружит эти действия. Формулировка проблемы остается неизменной.

Компьютерные системы и веб-сервисы постепенно централизуются, поэтому многие приложения предназначены для обслуживания миллионов и даже миллиардов пользователей. Объекты, которые становятся «повелителями информации», являются самой крупной мишенью для некорректного использования, но в то же время их положение весьма удачно для улучшения защиты данных и пользователей. С учетом появления мощных аппаратных средств обработки данных и разработки более эффективных алгоритмов анализа данных и машинного обучения именно сейчас наступило наилучшее время для применения потенциальных преимуществ машинного обучения в сфере обеспечения безопасности.

В этой книге мы показываем практическое применение методик машинного обучения и анализа данных в различных проблемных областях обеспечения

---

[1] К сожалению, информация по ссылке в оригинале **http://bit.ly/2EKGDDZ** недоступна (Page not found). – *Прим. перев.*

[2] К сожалению, информация по ссылке в оригинале **http://bit.ly/2DbwD66** недоступна (Page not found). – *Прим. перев.*

безопасности и защиты от некорректного использования. Подробно излагаются методы определения применимости разнообразных методик машинного обучения в разных ситуациях. Особое внимание сосредоточено на руководящих принципах, которые помогут использовать данные для улучшения защиты. Нашей целью не являются решения для каждой конкретной проблемы безопасности, с которой можно встретиться в реальной практике, – мы хотим предоставить читателю рабочую среду, основу для работы с данными и для решения задач защиты, а также комплект инструментов, из которого вы можете выбрать наиболее подходящее средство (метод) для решения конкретной поставленной задачи.

Следующая часть этой главы формирует контекст для всей книги в целом: рассматривается, каким опасностям подвергаются современные компьютерные и сетевые системы, что такое машинное обучение и как его можно применить для борьбы с вышеупомянутыми опасностями. В конце главы подробно описываются методы борьбы со спамом, которые представляют собой конкретный пример применения машинного обучения в сфере обеспечения безопасности, который можно обобщить для последующего применения практически в любой области.

Общий обзор потенциальных киберугроз

Общая картина, характеризующая нарушителей и злоумышленников в области компьютерной безопасности, изменяется со временем, но общие категории угроз и опасностей остаются неизменными. Для того чтобы сорвать планы атакующих злоумышленников, проводятся специальные исследования, поэтому всегда важно правильно классифицировать различные типы реально существующих атак. На рис. 1.1 изображено дерево систематической классификации (таксономии) существующих киберугроз (Cyber Threat Taxonomy)[1], на котором можно видеть связи между конкретными типами киберугроз и категориями, которые в некоторых случаях могут быть достаточно сложными.

Начнем с определения основных типов киберугроз, которые будут рассматриваться в следующих главах книги:

- вредоносное программное обеспечение (ПО) (malware) или вирус (virus) – программное обеспечение, специально предназначенное для нанесения ущерба или получения несанкционированного доступа к компьютерным системам (malware – malicious software);
- червь (worm) – автономная вредоносная программа, способная размножаться и копировать себя на другие компьютерные системы;
- троянская программа (trojan) – вредоносная программа, выдающая себя за одну из обычных программ, чтобы избежать обнаружения;
- программа-шпион (spyware) – вредоносная программа, установленная на компьютерной системе без разрешения и даже без ведома оператора/пользователя для шпионажа и сбора информации. К этой категории также относятся кейлоггеры;

[1] Источник: European CSIRT Network project's Security Incidents Taxonomy (**https://www.enisa.europa.eu/topics/threat-risk-management/threats-and-trends**).

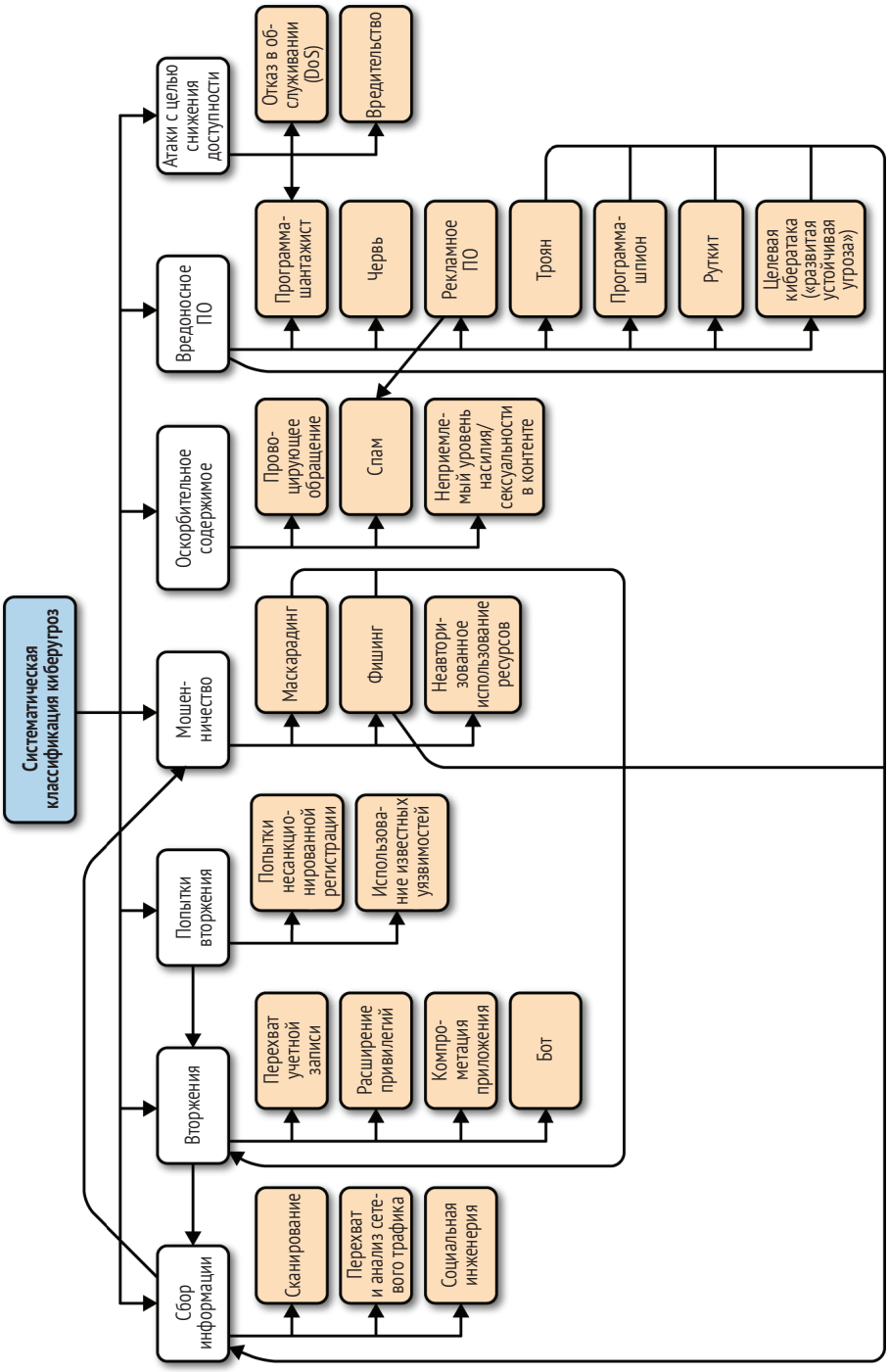


Рис. 1.1 ❖ Дерево систематической классификации (таксономии) существующих киберугроз (Cyber Threat Taxonomy)

- рекламное ПО (adware) – вредоносная программа, которая вводит непредусмотренные рекламные материалы (например, всплывающие окна, баннеры, видеоклипы) в подсистему пользовательского интерфейса, чаще всего появляющиеся при просмотре пользователем веб-контента;
- программа-шантажист (ransomware) – вредоносная программа, специально предназначенная для ограничения функциональных возможностей компьютерных систем до тех пор, пока не будет выплачена определенная денежная сумма (выкуп);
- руткит (rootkit) – комплект ПО низкого уровня (чаще всего), специально предназначенного для получения доступа или полного захвата управления компьютерной системой (root обозначает самый высокий уровень доступа и управления системой);
- бэкдор, или «черный ход» (backdoor) – преднамеренно созданная или оставленная лазейка («дыра»), размещенная на периметре защиты системы и позволяющая в будущем получить доступ в обход подсистемы внешней защиты;
- бот (bot) – вариант вредоносной программы, позволяющий атакующему в удаленном режиме перехватить управление компьютерными системами, превращая их в «зомби»;
- ботнет, сеть ботов (botnet) – крупная сеть ботов;
- эксплойт (exploit) – фрагмент кода или программа, использующая конкретные уязвимости в других прикладных программах или программных средах;
- сканирование (scanning): при этом типе атаки на компьютерные системы отправляются разнообразные запросы, часто в режиме простого перебора (грубой силы), с целью обнаружения слабых мест и уязвимостей, а также для сбора информации;
- перехват и анализ сетевого трафика (sniffing) – незаметное наблюдение и фиксация сетевого трафика и внутреннего трафика на сервере без ведома сетевых операторов;
- кейлоггер (keylogger) – деталь аппаратуры или фрагмент ПО (чаще всего скрытые от пользователя), которые фиксируют все нажатия клавиш на клавиатуре или действия на другом устройстве ввода;
- спам (spam) – незапрашиваемые сообщения, рассылаемые в крупных масштабах, чаще в рекламных целях. Обычно используется электронная почта, но спам также может распространяться в смс-сообщениях или через провайдера системы обмена сообщениями (например, WhatsApp);
- атака во время процедуры регистрации (login attack) – многочисленные, обычно автоматизированные попытки подобрать учетные данные для систем аутентификации, реализованные в форме простого перебора (грубой силы) или использующие похищенные/незаконно приобретенные учетные данные;
- захват учетной записи (account takeover – ATO) – получение доступа к чужой учетной записи, как правило, с целью нарушения коммерческой деятельности, кражи личных данных, похищения денежных средств и т. п. Обычно перехват учетной записи является целью атаки во время процедуры регистрации, но также может иметь меньший масштаб и более высокую целенаправленность (например, шпионское ПО, социальная инженерия);

- фишинг (phishing), или маскарадинг (masquerading) – установление связи от имени человека или организации, заслуживающих доверия. Цель: убедить объект фишинга предоставить личную информацию или передать права владения материальными ценностями;
- направленный, или целевой, фишинг (spear phishing) – фишинг, целью которого является конкретный пользователь, с использованием информации об этом пользователе, собранной из различных внешних источников;
- социальная инженерия (social engineering) – получение информации от людей с применением нетехнических методов, таких как ложная информация, обман, подкуп, шантаж и т. п.;
- провоцирующее обращение (incendiary speech) – унижающее, дискредитирующее или другое подобное враждебное обращение, адресованное отдельному лицу или группе лиц;
- атака типа «отказ в обслуживании», или DoS-атака, и распределенная DoS-атака – атаки, направленные на снижение доступности систем и выполняемые с помощью многочисленных некорректных запросов и/или запросов, содержащих большие объемы данных. Зачастую такие атаки также нарушают целостность и надежность систем;
- целевая кибератака («развитая устойчивая угроза») (advanced persistent threat – APT) – целенаправленная атака на сеть или на хост, при которой скрывающийся нарушитель остается необнаруженным в течение долгого времени и постоянно похищает и отслеживает передаваемые данные;
- уязвимость нулевого дня (zero-day vulnerability) – уязвимость или ошибка в ПО или в компьютерной системе, которая неизвестна производителю (поставщику), позволяющая воспользоваться ею (атака «нулевого дня»), прежде чем у производителя (поставщика) появится возможность устранить эту проблему.

Экономическая подоплека кибератак

По каким причинам предпринимаются кибератаки? Преступления в интернете становятся все более коммерциализированными по сравнению с начальным этапом распространения этой технологии. Переход от кибератак, мотивацией которых являлось зарабатывание определенной репутации (дешевая популярность, особенно в молодежной среде, известность и даже просто возможность совершать подобные проделки), к кибератакам с целью получения денег (прямые хищения денег, реклама, продажа личной секретной информации) стал весьма привлекательным процессом, особенно с точки зрения злоумышленников. В наши дни главная побуждающая причина кибератак – получение крупных денежных сумм. Атаки на финансовые организации или каналы (онлайновые платежные платформы, учетные записи, содержащие данные о кредитных и дебетовых картах, кошельки биткойнов и т. п.) могут открыть атакующим злоумышленникам прямой доступ к денежным средствам. Но с увеличением денежного объема, вовлеченного в онлайновый оборот, финансовые организации все чаще применяют усовершенствованные механизмы защиты, усложняющие жизнь злоумышленников. Из-за соблазна найти более короткий и легкий путь в сферу финансовой деятельности «рынок», предлагающий использование уязвимостей в системах защиты

финансовых организаций и каналов платежей, также представляет собой многочисленное и оживленное сообщество. Злоумышленники постоянно ищут объекты с более слабой защитой, неправильно эксплуатируемые системы с уязвимостями из-за ошибок при проектировании, а также обращаются к более изощренным методикам, которые в конечном итоге позволяют получить в свое распоряжение некоторую денежную сумму.

Рынок услуг взломщиков

Всем известно о существовании рынков darknet и не вполне законных форумов хакеров и взломщиков. До появления организованных подпольных сообществ, занимающихся незаконной деятельностью, только самые умелые хакеры способны были принять участие в организации кибератак и взломе учетных записей и компьютерных систем. Но с ростом коммерциализации хакерской деятельности и при массовом применении компьютеров во всех сферах жизни даже малоопытные «хакеры»[1] смогли внедриться в экосистему кибератак, получая в свое распоряжение (прибретая) информацию об уязвимостях и удобные для любого пользователя хакерские скрипты, программы и инструментальные средства для осуществления собственных кибератак.

На рынке уязвимостей нулевого дня существуют и практически законные, и абсолютно незаконные варианты. Торговля информацией об уязвимостях и методиками их использования может стать реальным источником дохода как для исследователей в области защиты и обеспечения безопасности, так и для хакеров. Но большинство самых «элитных» хакеров не склонно пользоваться уязвимостями «нулевого дня» и участвовать в организации массовых атак. Слишком велик риск, а кроме того, процесс обналичивания слишком долговременный и неопределенный. Создание ПО, которое дает возможность любому неопытному скрипт-кидди (script-kiddy) совершить попытку реального взлома, продажа информации об уязвимостях на свободных рынках, а в некоторых случаях даже небольшие компании, предоставляющие консультации и сервисы по взлому, – все это позволяет поверить в то, что существует быстрый и легкий путь к финансовому благополучию. Как во времена знаменитой золотой лихорадки в Калифорнии в конце 1840-х годов, продавцы, проявляющие чрезвычайную учтивость и любезность к растущей армии охотников за богатством, гораздо чаще получают неожиданные прибыли, чем сами охотники.

Косвенная монетизация

Процесс монетизации (или обналичивания денежных средств), предпринимаемый злоумышленниками, связан с различными типами компьютерных атак и настолько многообразен, что заслуживает подробнейшего изучения. Здесь мы не будем углубляться в исследования подобного рода, но рассмотрим несколько примеров, демонстрирующих, как может осуществляться косвенная монетизация.

Распространение вредоносного ПО было коммерциализировано способом, похожим на развитие облачных вычислений (cloud computing) и развертывание

[1] Вообще говоря, они не заслуживают звания «хакер» в изначальном, не столь криминализированном значении этого термина. – *Прим. перев.*

провайдеров инфраструктуры как сервиса (IaaS – Infrastructure-as-a-Service). Рыночные отношения типа «плата за установку» (pay-per-install – PPI) при распространении вредоносного ПО представляют собой вполне сложившуюся сложную экосистему, предоставляющую мощные каналы распространения, доступные и авторам, и потребителям[1]. Аренда ботнета основана на том же принципе, что и облачная инфраструктура, предоставляемая по запросу, с почасовой оплатой выделяемых ресурсов за приемлемую цену. Развертывание вредоносного ПО на удаленных серверах также может быть прибыльным с финансовой точки зрения, с разнообразными специфическими способами монетизации. Целевые атаки на конкретные объекты иногда связаны с получением какой-либо финансовой выгоды, а распространение программ-шантажистов может быть достаточно эффективным способом вымогательства денежных средств у обширной группы жертв.

Шпионское ПО может способствовать похищению личной секретной информации, которую затем можно выгодно продать оптом на тех же онлайновых рынках ПО для шпионажа. Рекламное ПО и средства распространения спама можно использовать как дешевый способ рекламирования не вполне легальных фармацевтических товаров и финансовых инструментов. Онлайновые учетные записи часто перехватываются с целью похищения ценностей, хранящихся в особой форме, как, например, подарочные (призовые) карты, бонусные баллы за лояльность, открытые кредиты в магазинах или премиальный возврат денег при покупках. Похищенные номера кредитных карт, номеров социальной страховки, учетных записей электронной почты, номеров телефонов, адресов и прочая личная секретная информация может быть продана в режиме онлайн преступникам, намеревающимся заняться воровством, подделками, мошенничествами и прочими подобными деяниями. Но процесс монетизации (обналичивания), в особенности если злоумышленник располагает номером кредитной карты жертвы, может стать долгим и сложным. Из-за возможности легкого похищения такой информации компании, предоставляющие кредитные карты, а также компании, обслуживающие учетные записи для спецхранения ценностей, часто применяют хитроумные технические методики для предотвращения монетизации, предпринимаемой злоумышленниками. Например, если возникает подозрение, что учетные записи скомпрометированы, то они объявляются некорректными и неработающими, а для карт с премиальным возвратом денег требуются дополнительные процедуры аутентификации.

Подведем итоги

Побудительные мотивы киберпреступников сложны, а способы монетизации извилисты. Тем не менее финансовые выгоды от атак в интернете могут стать мощным стимулом для технически подготовленных людей, особенно из не очень богатых стран и сообществ. Пока компьютерные атаки способны создавать обширную криминальную сферу деятельности для злоумышленников, такие атаки будут продолжаться.

---

[1] *Juan Caballero et al*. Measuring Pay-per-Install: The Commoditization of Malware Distribution. Proceedings of the 20th USENIX Conference on Security (2011).

Что такое машинное обучение

В самом начале «технологической эры» ученые мечтали о том, чтобы научить компьютеры рассуждать логически и принимать «разумные» решения точно так же, как это делает человек, выводя общие правила и выделяя концепции из больших объемов сложной информации без точно определенных инструкций.

Машинное обучение (machine learning) связано только с одной из этих перспектив, а именно с алгоритмами и процессами, которые являются «обучающими» в смысле обеспечения возможности обобщать данные и практические сведения, полученные в прошлом, для того чтобы предсказывать будущие результаты. По своей сущности машинное обучение представляет собой набор математических методов, реализованных в компьютерных системах, обеспечивающих процесс извлечения информации, обнаружение шаблонов и формирование выводов из данных.

На самом высоком (обобщенном) уровне при машинном обучении с учителем (supervised machine learning)[1] применяется методика Байеса (Bayes) для выявления знаний, использующая известные вероятности наступления ранее наблюдаемых событий для определения вероятностей новых событий. Машинное обучение без учителя (unsupervised machine learning)[2] выделяет абстрактные признаки из наборов непомеченных данных и применяет эти признаки к новым данным. Обе группы методик можно применить к задачам классификации (classification), т. е. распределения наблюдений по категориям, или регрессии (regression), т. е. прогнозирование числовых характеристик наблюдения.

Предположим, что необходимо выполнить классификацию группы животных, разделяя их на млекопитающих и рептилий. По методике обучения с учителем берется группа животных, для которых явно указана их категория (например, четко определено, что собака и слон – млекопитающие, а аллигатор и игуана – рептилии). Затем мы пытаемся извлечь какие-либо характерные признаки из каждого элемента этих помеченных данных и найти сходство в данных признаках, позволяющее различать животных, принадлежащих к разным классам. Например, очевидно, что собака и слон порождают живое потомство, в отличие от аллигатора и игуаны. Бинарное свойство «порождает живое потомство» называют характеристикой или признаком (feature), т. е. полезной абстракцией для наблюдаемых признаков, которые позволяют сравнивать различные наблюдения. После окончательного определения набора характеристик, которые могут помочь отличить млекопитающих от рептилий в помеченных данных, можно начать выполнение алгоритма обучения на наборе помеченных данных, затем применить то, чему научился алгоритм, к новым, ранее не называемым животным. Если применить этот алгоритм к сурикату, то после обучения должна быть выполнена классификация, относящая это животное либо к млекопитающим, либо к рептилиям. Получив набор характеристик из данных об этом новом животном, алгоритм знает, что сурикат не откладывает яиц, не имеет чешуи и теплокровный. На основании

---

[1] Также используются термины «контролируемое обучение» и «управляемое обучение». – *Прим. перев.*

[2] Также используются термины «неконтролируемое обучение» и «самообучение». – *Прим. перев.*

предыдущих наблюдений делается вывод: сурикат принадлежит к категории млекопитающих, и этот вывод абсолютно верный.

По методике обучения без учителя предпосылка та же самая, но алгоритм не получает в свое распоряжение начальный набор помеченных данных о животных. Вместо этого алгоритм должен группировать различные наборы элементов данных таким способом, чтобы в результате получить бинарную классификацию. Определив по наблюдениям, что большинство животных, которые не имеют чешуи, порождают живое потомство и являются теплокровными, а большинство животных, которые имеют чешую, откладывают яйца и являются холоднокровными, алгоритм способен распределять по двум категориям животных из предоставленной группы и предсказывать основные характеристики так же, как и в случае обучения с учителем.

Алгоритмы машинного обучения основаны на математике и статистике, а алгоритмы, выявляющие шаблоны, корреляции и аномалии в данных, имеют различные степени сложности. В следующих главах будут более подробно рассматриваться механизмы некоторых наиболее часто применяемых алгоритмов машинного обучения, используемых в этой книге. Книга не поможет вам в полной мере освоить машинное обучение, в ней не излагаются подробно математические и теоретические основы этой технологии. Авторы попытались привить читателю разумное отношение к машинному обучению и практические навыки для проектирования и реализации «интеллектуальных» динамически адаптируемых систем в контексте обеспечения безопасности.

Чем не является машинное обучение

Искусственный интеллект (artificial intelligence – AI) – широко распространенный, но весьма неопределенный термин, который в общем обозначает алгоритмические решения сложных задач, которые обычно решаются людьми. Как показано на рис. 1.2, машинное обучение является основным конструктивным блоком (и даже ядром) искусственного интеллекта. Например, автомобили с автоматическим управлением обязательно должны классифицировать наблюдаемые образы-объекты как людей, автомобили, деревья и т. п., а кроме того, непременно должны прогнозировать положение и скорость других машин. Они также должны определять угол поворота колес для изменения направления движения. Такие задачи классификации и прогнозирования решаются с использованием машинного обучения, поэтому система автоматического управления автомобилем представляет собой некоторую форму искусственного интеллекта. Существуют и другие элементы механизма принятия решения в интеллектуальной системе автоматического управления, которые жестко закодированы в виде набора правил, следовательно, их нельзя считать методами машинного обучения. Машинное обучение помогает создавать искусственный интеллект, но это не единственный метод формирования ИИ.

Глубокое обучение (deep learning) – это еще один широко распространенный термин, который, вообще говоря, тесно связан с машинным обучением. Глубокое обучение представляет собой строго ограниченное подмножество методик машинного обучения, применяемых к особому классу многоуровневых моделей, в которых используются уровни упрощенных статистических компонентов для изучения представлений данных. Нейронная сеть (neural network) – более общий

термин для обозначения этого типа многоуровневой статистической архитектуры обучения, которая может быть или не быть «глубокой» (т. е. иметь или не иметь много уровней). Эта тема превосходно раскрыта в книге Deep Learning Иэна Гудфеллоу (Ian Goodfellow), Йошуа Бенджио (Yoshua Bengio) и Аарона Курвиля (Aaron Courville), выпущенной издательством MIT Press.

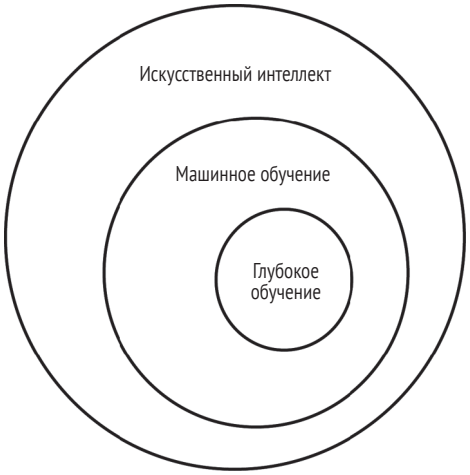


Рис. 1.2 ❖ Взаимосвязь искусственного интеллекта с машинным обучением и глубоким обучением

Статистический анализ является важнейшей частью машинного обучения: результаты выполнения алгоритмов машинного обучения часто представлены в форме вероятностей и доверительных интервалов. Некоторые статистические методы будут кратко описаны при рассмотрении темы выявления аномалий, но мы оставили за пределами данной книги многие вопросы, касающиеся методов проверки экспериментальных и статистических предположений. Эти вопросы в полной мере рассматриваются в книге Probability & Statistics for Engineers & Scientists, Роналд Уолпол (Ronald Walpol) и др. (издательство Prentice Hall).

Что такое искусственный интеллект

Определение искусственного интеллекта представляет собой несколько более спорную тему, по сравнению с определением машинного обучения. Машинное обучение – это комплект алгоритмов статистического обучения, способных создать обобщенные абстракции (модели) посредством наблюдения и тщательного анализа наборов данных. Системы искусственного интеллекта были определены менее точно как механизмы управляемого машиной принятия решений, которые могут достигать уровня интеллекта, сравнимого с человеческим. Но в какой степени этот интеллект должен быть «сравнимым» с человеческим разумом, чтобы можно было считать его искусственным интеллектом? Вполне объяснимо, что разнообразие предположений и трактований этого термина чрезвычайно затрудняет определение границ данной области, которое стало бы общепринятым.

Другие варианты использования машинного обучения

Следует отметить, что нет препятствий для использования преимуществ машинного обучения злоумышленниками, чтобы избежать обнаружения и обойти систему защиты. Защищающаяся сторона имеет возможность извлекать полезный опыт из атак и соответствующим образом принимать контрмеры, но и атакующая сторона также изучает внутренние механизмы систем защиты, чтобы извлечь личную выгоду. Распространители спама освоили практическое применение полиморфизма (т. е. изменение внешнего вида содержимого без изменения его смысла), для того чтобы не позволить распознать откровенно их рекламный контент, или для прощупывания фильтров спама с выполнением A/B-тестирования содержимого электронной почты и изучением процентного соотношения успешных и отфильтрованных вариантов. Обе стороны используют машинное обучение в своих «кампаниях фаззинг-тестирования, направленных на ускорение процесса поиска уязвимостей в программном обеспечении» (**http://www.vdiscover.org/OS-fuzzing.html**). Злоумышленники могут воспользоваться машинным обучением даже для исследования ваших личных данных и интересов в соцсетях, чтобы сформировать идеальное фишинговое сообщение персонально для вас.

Наконец, использование динамических и адаптивных методов в области обеспечения безопасности всегда связано с определенной долей риска. Особенно в тех случаях, когда обоснованность прогнозов машинного обучения часто становится недостаточной, атакующая сторона узнает о различных алгоритмах, ставших причиной ошибочных предсказаний или неправильного обучения[1]. В этой постоянно расширяющейся области обучения, называемой «состязательным машинным обучением» (adversarial machine learning), атакующие с той или иной степенью доступа к системе машинного обучения могут выполнять некоторый набор атак для достижения своих целей. В главе 8 подробно рассматривается эта тема и читателю предоставляется более полное описание задач и решений в этой области.

Алгоритмы машинного обучения во многих случаях не предназначены специально для обеспечения безопасности, поэтому зачастую уязвимы для попыток вмешательства в их работу, предпринимаемых мотивированным злоумышленником. Поэтому важно иметь полное представление о подобных моделях угроз при проектировании и создании систем машинного обучения, специализированных для обеспечения безопасности.

Практические варианты использования машинного обучения для обеспечения безопасности

В этой книге рассматриваются различные приложения, предназначенные для обеспечения компьютерной безопасности, для которых машинное обучение уже продемонстрировало многообещающие результаты. Практическое применение машинного обучения и науки о данных (даталогии) для решения задач – дело непростое. Несмотря на то что удобные в использовании программные библиотеки снижают уровень сложности, для разработчиков сохраняется необходимость постоянно принимать множество самостоятельных решений.

---

[1] *Ling Huang et al.* Adversarial Machine Learning. Proceedings of the 4th ACM Workshop on Artiicial Intelligence and Security (2011): 43–58.

Рассматривая различные примеры в текущей главе, мы исследуем наиболее часто возникающие в реальной практике проблемы при проектировании систем машинного обучения, вне зависимости от того, предназначены они для обеспечения безопасности или нет. Приложения, описываемые в данной книге, не новы, поэтому можно обнаружить обсуждаемые здесь методики даталогии во многих компьютерных системах, с которыми вы, возможно, ежедневно имеете дело.

Практические варианты использования машинного обучения в обеспечении безопасности можно классифицировать по двум основным категориям: распознавание шаблонов (pattern recognition) и выявление аномалий (anomaly detection). Границу между распознаванием шаблонов и выявлением аномалий не всегда можно четко определить, но для каждой конкретной задачи поставлена точно сформулированная цель. При распознавании шаблонов мы пытаемся обнаружить явные или неявные характеристики, скрытые в данных. Эти характеристики, выделенные и объединенные в наборы признаков, могут использоваться для обучения алгоритма распознаванию других форм данных с точно таким же набором характеристик. Выявление аномалий – это получение знаний при противоположном подходе к той же задаче. Вместо изучения характерных шаблонов, существующих в конкретных наборах данных, главной целью становится определение понятия нормальности, которое описывает большую часть (например, более 95 %) данных в исследуемом наборе. После этого любые отклонения от установленной нормальности будут определяться как аномалии.

Распространено ошибочное мнение о том, что выявление аномалий является процессом распознавания набора «нормальных» шаблонов и установления их отличий от набора «ненормальных» шаблонов. Шаблоны, извлекаемые по методике распознавания шаблонов, непременно должны быть производными от исследуемых данных, используемых для предварительной подготовки (тренировки) алгоритма. С другой стороны, при использовании методики выявления аномалий возможно существование бесконечного количества аномальных шаблонов с характеристиками, соответствующими заявленным описаниям промахов (выбросов), даже тех, которые являются производными от гипотетических данных, реально не существующих в тренировочных или тестовых наборах данных.

Выявление спама, возможно, представляет собой классический пример распознавания шаблонов, поскольку спам обычно обладает вполне предсказуемым набором характеристик, и алгоритм можно подготовить для распознавания этих характеристик как шаблона, по которому классифицируются сообщения электронной почты. Кроме того, также допустима трактовка выявления спама как задачи выявления аномалий. Если есть возможность вывода набора признаков, которые описывают обычный сетевой трафик достаточно подробно и точно, для того чтобы определить существенные отклонения от нормы как спам, то задача решена успешно. Но в действительности задача выявления спама не вполне соответствует парадигме выявления аномалий, поскольку не составляет никакого труда убедиться в том, что в большинстве контекстов легче найти одинаковые свойства, присущие спам-сообщениям, чем в более обширном наборе данных обычного трафика.

Выявление вредоносного ПО и обнаружение ботнетов представляют собой другие типы приложений, которые явно попадают в категорию распознавания шаблонов, где машинное обучение становится особенно полезным при атаках с использованием полиморфизма, чтобы скрыться от обнаружения. Фаззинг (fuz-

zing) – это процесс передачи случайных входных данных в программу или какой-либо компонент программы с целью перевода приложения в непредусмотренное некорректное состояние. Чаще всего в таком процессе ставится задача достижения аварийного завершения программы или создания уязвимого режима ее работы с возможностью дальнейшего использования этой уязвимости. Плохо подготовленные («наивные») мероприятия по фаззинг-тестированию часто сводятся к простому итеративному проходу по трудно определяемому огромному пространству состояний приложения. Наиболее часто применяемое ПО для фаззинг-тестирования предоставляет средства оптимизации, которые существенно повышают эффективность этой методики по сравнению со «слепым» перебором состояний (**http://lcamtuf.coredump.cx/afl/**). В оптимизациях подобного рода также применялось и применяется машинное обучение – исследование образцов (шаблонов) ранее обнаруженных уязвимостей в похожих программах и ориентирование фаззера (fuzzer; программа фаззинг-тестирования) по маршруту аналогичного уязвимого кода или по характерным особенностям кода для потенциально более быстрого получения результатов.

При аутентификации пользователей и анализе поведения различия между распознаванием шаблонов и выявлением аномалий становятся менее очевидными. В случаях, когда модель угрозы хорошо известна, возможно, более подходящим является методика решения задачи с помощью двунаправленной трансформации данных при распознавании шаблонов (lens of pattern recognition). В других ситуациях более предпочтительно применение выявления аномалий. Во многих случаях система может применять обе методики для обеспечения наилучшего покрытия. Выявление промахов в сетевой среде – классический пример выявления аномалий, поскольку почти весь сетевой трафик строго соблюдает протоколы и обычное поведение соответствует набору шаблонов по форме или по порядку следования. В сети любая злонамеренная деятельность, которая не применяет технику маскарадинга для имитации обычного трафика, будет обнаружена с помощью алгоритмов выявления промахов. Решения других задач выявления событий, связанных с сетью, такие как выявление вредоносных URL, также могут рассматриваться с точки зрения выявления аномалий.

Управление доступом (access control) обозначает любой набор стратегий, управляющих возможностью пользователей какой-либо системы получать доступ к конкретным элементам информации. Часто используемые для защиты важной секретной информации от нежелательного раскрытия, стратегии управления доступом в большинстве случаев представляют собой первую линию защиты от проникновений и похищения информации. Технология машинного обучения постепенно нашла свое место в решениях задач управления доступом как средство облегчения жизни пользователей систем, находящихся во власти строгих и беспощадных стратегий управления доступом[1]. С помощью сочетания методики обучения без учителя и выявления аномалий такие системы могут делать логические выводы относительно информации о шаблонах доступа для конкретных пользователей или ролей в организации и принимать ответные меры при обнаружении несоответствия установленным шаблонам.

---

[1] *Evan Martin and Tao Xie.* Inferring Access-Control Policy Properties via Machine Learning. Proceedings of the 7th IEEE International Workshop on Policies for Distributed Systems and Networks (2006): 235–238.

Например, предположим, что существует система хранения записей о пациентах больницы, к которой необходимо обеспечить постоянный доступ медсестер и медтехников, но при этом они не должны устанавливать какие-либо связи и отношения между различными пациентами. С другой стороны, врачам разрешается делать запросы и объединять регистрационные записи группы пациентов для выявления аналогичных симптомов и диагнозов. Нет необходимости запрещать медсестрам и медтехникам выполнять запросы на получение записей о нескольких пациентах, потому что в редких случаях потребуется разрешение на подобные действия. Основанная на правилах строгая система управления доступом не сможет обеспечить гибкость и адаптируемость, которую способна предоставить методика машинного обучения.

В следующих главах книги будет более подробно рассматриваться выбор подобных приложений в реальной практике. Мы получим возможность обсудить нюансы, касающиеся применения машинного обучения для распознавания шаблонов (образов) и выявления аномалий в области обеспечения безопасности. В заключительной части текущей главы мы сосредоточимся на разборе примера борьбы со спамом как одной из наглядных иллюстраций применения базовых принципов, используемых в любом приложении машинного обучения для обеспечения безопасности.

Борьба со спамом: итеративный подход

Как уже было отмечено выше, пример борьбы со спамом является едва ли не самой старой задачей обеспечения компьютерной безопасности, которая к тому же успешно решается с помощью машинного обучения. В этом разделе подробно рассматривается эта тема и наглядно демонстрируется постепенное создание «интеллектуальной» системы классификации спама с использованием машинного обучения. Описываемый здесь подход является обобщенным для многих других типов задач обеспечения безопасности, включая и задачи, рассматриваемые в последующих главах. Но не следует полагать, что круг решаемых с помощью этого подхода задач ограничен лишь описанными в нашей книге, на самом деле он более широк.

Рассмотрим случай, когда предлагается решить задачу устранения угрозы распространения по электронной почте спама, мешающего работе сотрудников в некоторой организации. По определенным причинам получено распоряжение разработать собственное решение, а не использовать стороннее коммерческое ПО. Обладая правами доступа администратора на внутренних серверах электронной почты в организации, вы можете извлекать тело (содержимое) e-mail-сообщений для анализа. Все сообщения электронной почты помечаются получателями соответственно либо как «спам» (spam), либо как «не спам» (non-spam; «ham»)[1], поэтому не нужно тратить времени на фильтрацию данных[2].

---

[1] Первоначально слово «SPAM» появилось в 1936 г. как аббревиатура от «**Sp**iced h**am**» (острая ветчина) и было торговой маркой для мясных консервов компании Hormel Foods Corporation – острого колбасного фарша из свинины. Всемирную известность в применении к назойливой рекламе термин «SPAM» получил благодаря знаменитому скетчу «Спам» из известного телевизионного шоу «Летающий цирк Монти Пайтона» (1969). – *Прим. перев.*

[2] В реальной практике приходится затрачивать достаточно много времени на фильтрацию данных, чтобы сделать их доступными и действительно полезными для применяемых алгоритмов.

Человек успешно распознает спам, поэтому начнем с реализации простого решения, приближенно имитирующего процесс человеческого мышления при выполнении этой задачи. Теоретическая предпосылка заключается в наличии или отсутствии некоторых известных ключевых слов в сообщении электронной почты – это четкий признак того, что сообщение является спамом или не спамом. Например, замечено, что слово «лотерея» (lottery) весьма часто встречается в спам-сообщениях, но крайне редко появляется в обычных деловых письмах. Возможно, вы в итоге составите список таких слов и выполните классификацию, проверяя, содержится ли в тексте сообщения какое-либо слово из этого «черного списка».

Набор данных, используемый для решения поставленной задачи, взят из источника 2007 TREC Public Spam Corpus (**https://plg.uwaterloo.ca/~gvcormac/treccorpus07/**). Это слегка отфильтрованный массив настоящих сообщений электронной почты, содержащий 75 419 экземпляров, собранных на сервере электронной почты в течение трех месяцев 2007 года. Треть этого набора данных составляют образцы спама, остальные – обычные сообщения. Набор данных создан участниками конференции Text REtrieval Conference (TREC) Spam Track (**https://trec.nist.gov/data/spam.html**) в 2007 году как часть работы по расширению границ самой передовой технологии выявления спама.

Для оценки эффективности работы различных методик будет применяться простой процесс проверки (валидации)[1]. Основной набор данных разделяется на неперекрывающиеся подготовительный (тренировочный) и тестовый наборы, при этом тренировочный набор состоит из 70 % данных (пропорция выбрана произвольно), а на долю тестового набора остается 30 % данных. Такая методика является стандартной практикой для оценки эффективности алгоритма или модели, разрабатываемых на основе подготовительного (тренировочного) набора данных. В дальнейшем проверенный алгоритм (или модель) обобщается для работы с независимым набором данных.

Первый этап – использование инструментального пакета Natural Language Toolkit (NLTK) (**http://www.nltk.org/**) для удаления морфологических аффиксов из слов для более адаптивного процесса поиска совпадений (такой процесс называют стеммингом (stemming), т. е. определением основы слова). Например, этот подход позволяет сократить слова «congratulations» и «congrats» до одной и той же основы «congrat». Также удаляются шумовые слова (stopwords) (например, артикли the и a, глаголы-связки is и are) перед процессом извлечения токенов-образ-

[1] Этот процесс валидации, который иногда называют условной валидацией (conventional validation), является не столь строгим методом, как перекрестная валидация (cross-validation), обозначающая целый класс методов, многократно генерирующих все (или большинство) различные возможные варианты разделения набора данных (на подготовительный и тестовый наборы) и выполняющих проверку (валидацию) прогнозирующего алгоритма машинного обучения отдельно на каждом из этих наборов. Результатом перекрестной валидации является усредненная точность прогнозов по всем этим различным вариантам разделенных наборов. Перекрестная валидация оценивает точность модели лучше, чем условная валидация, поскольку позволяет избежать возможных потерь информации при обработке единственного подготовительного/тестового варианта набора данных, который может не вполне корректно определить статистические свойства данных (обычно это не становится причиной для беспокойства, если подготовительный (тренировочный) набор данных достаточно велик). В нашем примере для простоты выбрана методика условной валидации.

цов, поскольку шумовые слова обычно не несут какой-либо смысловой нагрузки. Определяется набор вспомогательных функций[1] для загрузки и предварительной обработки данных и меток, как показано в следующем коде[2]:

```
import string
import e-mail
import nltk

punctuations = list(string.punctuation)
stopwords = set(nltk.corpus.stopwords.words('english'))
stemmer = nltk.PorterStemmer()

# Объединение различных частей e-mail-сообщения в простой список строк
def flatten_to_string(parts):
    ret = []
    if type(parts) == str:
        ret.append(parts)
    elif type(parts) == list:
        for part in parts:
            ret += flatten_to_string(part)
    elif parts.get_content_type == 'text/plain':
        ret += parts.get_payload()
    return ret

# Извлечение текста темы и тела из одного e-mail-файла
def extract_e-mail_text(path):
    # Загрузка одного e-mail-сообщения из входного файла
    with open(path, errors='ignore') as f:
        msg = e-mail.message_from_file(f)
    if not msg:
        return ""

    # Чтение темы сообщения
    subject = msg['Subject']
    if not subject:
        subject = ""

    # Чтение тела сообщения
    body = ' '.join(m for m in flatten_to_string(msg.get_payload())
                    if type(m) == str)
    if not body:
        body = ""
    return subject + ' ' + body

# Обработка одного e-mail-файла для преобразования слов в основы-токены
def load(path):
    e-mail_text = extract_e-mail_text(path)
    if not e-mail_text:
```

[1] Эти вспомогательные функции определены в файле *chapter1/e-mail_read_util.py* (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter1/e-mail_read_util.py**) в репозитории кода для данной книги.

[2] Для выполнения этого кода необходимо установить Punkt Tokenizer Models и массив шумовых слов в NLTK с помощью утилиты `nltk.download()`.

```
        return []

    # Разбивка сообщения на токены
    tokens = nltk.word_tokenize(e-mail_text)

    # Удаление знаков пунктуации из токенов
    tokens = [i.strip("".join(punctuations)) for i in tokens
              if i not in punctuations]

    # Удаление шумовых слов и выделение основ токенов
    if len(tokens) > 2:
        return [stemmer.stem(w) for w in tokens if w not in stopwords]
    return []
```

Далее загружаются сообщения электронной почты и метки. В этом наборе данных каждое сообщение помещено в отдельный файл (*inmail.1*, *inmail.2*, *inmail.3*, ...), кроме того, имеется отдельный файл меток (*full/index*) в следующем формате:

```
spam   ../data/inmail.1
ham    ../data/inmail.2
spam   ../data/inmail.3
...
```

В файле меток в начале каждой строки содержится метка «spam» или «ham» для всех образцов сообщений в исследуемом наборе данных. Этот набор данных считывается, и формируется черный список слов, характеризующих спам[1]:

```
import os

DATA_DIR = 'datasets/trec07p/data/'
LABELS_FILE = 'datasets/trec07p/full/index'
TRAINING_SET_RATIO = 0.7

labels = {}
spam_words = set()
ham_words = set()

# Чтение меток
with open(LABELS_FILE) as f:
    for line in f:
        line = line.strip()
        label, key = line.split()
        labels[key.split('/')[-1]] = 1 if label.lower() == 'ham' else 0

# Разделение массива на тренировочный и тестовый наборы данных
filelist = os.listdir(DATA_DIR)
X_train = filelist[:int(len(filelist)*TRAINING_SET_RATIO)]
X_test = filelist[int(len(filelist)*TRAINING_SET_RATIO):]

for filename in X_train:
    path = os.path.join(DATA_DIR, filename)
    if filename in labels:
```

[1] Этот пример можно найти в файле примечаний Python Jupyter *chapter1/spam-fighting-blacklist.ipynb* в репозитории исходного кода для данной книги (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter1/spam-fighting-blacklist.ipynb**).

```
        label = labels[filename]
        stems = load(path)
        if not stems:
            continue
        if label == 1:
            ham_words.update(stems)
        elif label == 0:
            spam_words.update(stems)
        else:
            continue

blacklist = spam_words - ham_words
```

При внимательном изучении токенов в черном списке `blacklist` можно заметить, что многие слова не несут смысловой нагрузки (например, слова в кодировке Unicode, URL, имена файлов, символы, иностранные слова). Эту проблему можно устранить с помощью более тщательной фильтрации данных, но даже такой упрощенный результат должен в достаточно неплохой степени соответствовать целям нашего эксперимента:

```
greenback, gonorrhea, lecher, ...
```

Оценивая работу применяемой методики на 22 626 сообщениях электронной почты из тестового набора, мы обнаруживаем, что этот упрощенный алгоритм не так эффективен, как можно было надеяться. Результаты объединены в итоговом отчете в форме матрицы неточностей или несоответствий (confusion matrix) размером 2×2, где указано количество образцов, для которых были сделаны прогнозы, и предварительно присвоенные метки для каждого из четырех возможных вариантов (табл. 1.1).

Таблица 1.1

	Прогнозируемый HAM	**Прогнозируемый SPAM**
Действительный HAM	6772	714
Действительный SPAM	5835	7543
Истинно положительное срабатывание: прогнозируемый спам + действительный не спам Истинно отрицательное срабатывание: прогнозируемый не спам + действительный не спам Ложноположительное срабатывание: прогнозируемый спам + действительный не спам Ложноотрицательное срабатывание: прогнозируемый не спам + действительный спам		

Процентные соотношения полученных результатов показаны в табл. 1.2.

Таблица 1.2

	Прогнозируемый HAM	**Прогнозируемый SPAM**
Действительный HAM	32,5 %	3,4 %
Действительный SPAM	28,0 %	36,2 %
Точность классификации: 68,7 %		

Если не обращать внимания на тот факт, что 5,8 % e-mail-сообщений не были классифицированы из-за ошибок предварительной обработки, то можно отметить, что реальная эффективность этого простейшего алгоритма вполне неплоха. Примененная методика черного списка спам-слов дает точность классификации 68,7 % (т. е. общее количество в процентах правильных меток). Но в этот черный список не включены многие слова, которые используются в спам-сообщениях, поскольку те же слова часто встречаются и в обычных сообщениях. Кроме того, непрерывная поддержка постоянно изменяющегося набора слов, по которым можно четко определять спам и нормальные тексты, кажется почти невыполнимой задачей. Возможно, сейчас нужно опять вернуться на этап предварительного проектирования.

Вероятно, вы помните, что в начале главы упоминался один из часто применяемых старых способов борьбы e-mail-провайдеров со спамом – нечеткое хеширование спам-сообщений и фильтрация сообщений, для которых вычислено похожее хеш-значение. Это некоторый тип совместной фильтрации (collaborative filtering), который полагается на здравый смысл других пользователей исследуемой платформы для формирования «коллективного разума», способного, как можно надеяться, правильно обобщать и идентифицировать новый приходящий спам. Здесь делается предположение о том, что при создании спама используются некоторые автоматические средства, следовательно, генерируемые спам-сообщения будут лишь незначительно отличаться друг от друга. Алгоритм нечеткого хеширования, или более точно – чувствительное к местоположению хеширование (locality-sensitive hash – LSH), может позволить находить приблизительные совпадения сообщений электронной почты, которые были помечены как спам.

После ряда исследований и изысканий обнаруживается datasketch (**https://github.com/ekzhu/datasketch**), полнофункциональный пакет на языке Python, содержащий эффективные реализации алгоритма MinHash + LSH[1] для выполнения операции сравнения строк с оценкой издержек на сублинейный запрос (sublinear query cost) (с учетом мощности множества спам-признаков). MinHash выполняет преобразование наборов строковых токенов в короткие сигнатуры, сохраняя при этом характерные свойства первоначальных входных данных, которые позволяют обнаруживать соответствующие совпадения. Затем LSH можно применять к сигнатурам MinHash вместо необработанных токенов, что значительно повышает эффективность. MinHash обеспечивает преимущества в производительности за счет некоторых потерь точности, поэтому в итоговых результатах возможно возникновение ложноположительных и ложноотрицательных срабатываний. Тем не менее выполнение простейшего нечеткого сравнения строк в каждом сообщении электронной почты со всеми n спам-сообщениями из тренировочного набора имеет сложность запроса O(n) (если каждый раз сканируется весь массив)

[1] Подробнее см. главу 3 книги Mining of Massive Datasets, 2nd ed., by Jure Leskovec, Anand Rajaraman and Jeffrey David Ullman (Cambridge University Press).

или O(n) сложность по памяти (если сформирована хеш-таблица для всего массива). Вы сами должны принять решение, какой вариант является более приемлемым[1]:

```
from datasketch import MinHash, MinHashLSH

# Извлечение только файлов со спамом для вставки в детектор совпадений LSH
spam_files = [x for x in X_train if labels[x] == 0]

# Инициализация детектора совпадений MinHashLSH с помощью порогового значения
# меры сходства Жаккарда, равного 0.5, и 128 функций перестановки MinHash
lsh = MinHashLSH(threshold=0.5, num_perm=128)

# Заполнение детектора совпадений LSH значениями тренировочного набора спам-данных MinHash
for idx, f in enumerate(spam_files):
    minhash = MinHash(num_perm=128)
    stems = load(os.path.join(DATA_DIR, f))
    if len(stems) &lt; 2: continue
    for s in stems:
        minhash.update(s.encode('utf-8'))
    lsh.insert(f, minhash)
```

Теперь необходимо присвоить прогнозируемые метки детектора совпадений LSH данным тестового набора:

```
def lsh_predict_label(stems):
    '''
    Queries the LSH matcher and returns:
        0 if predicted spam
        1 if predicted ham
       -1 if parsing error
    '''
    minhash = MinHash(num_perm=128)
    if len(stems) < 2:
        return -1
    for s in stems:
        minhash.update(s.encode('utf-8'))
    matches = lsh.query(minhash)
    if matches:
```

[1] Отметим, что в рассматриваемом примере для параметра объекта `MinHashLSH threshold` задано значение `0.5`. Эта конкретная реализация LSH использует меры сходства или коэффициенты Жаккара (Jaccard similarities) между значениями MinHash в наборе данных и запросом MinHash, возвращая список объектов, соответствующих пороговому условию (т. е. мере сходства Жаккара > 0.5). Алгоритм MinHash генерирует короткие неповторяющиеся сигнатуры для строк, передавая случайные перестановки строк через функцию хеширования. Значение параметра конфигурации `num_perm` 128 означает, что 128 случайных перестановок текущей строки вычисляются и передаются через функцию хеширования. Вообще говоря, чем больше случайных перестановок используется в алгоритме, тем более точно вычисляется хеш-значение.
Этот пример можно найти в файле примечаний Python Jupyter *chapter1/spam-fighting-lsh.ipynb* в репозитории исходного кода для данной книги (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter1/spam-fighting-lsh.ipynb**).

```
        return 0
    else:
        return 1
```

После выполнения программы можно видеть следующие результаты (табл. 1.3).

Таблица 1.3

	Прогнозируемый HAM	**Прогнозируемый SPAM**
Действительный HAM	7350	136
Действительный SPAM	2241	11 038

Те же результаты в процентном выражении показаны в табл. 1.4.

Таблица 1.4

	Прогнозируемый HAM	**Прогнозируемый SPAM**
Действительный HAM	35,4 %	0,7 %
Действительный SPAM	10,8 %	53,2 %
Точность классификации: 88,6 %		

Это приблизительно на 20 % лучше, чем при использовании методики простейшего черного списка, и значительно лучше по критерию ложноположительных срабатываний (т. е. прогнозируемый спам + действительный не спам). Но эти результаты все равно уступают результатам применения современных фильтров спама. При тщательном исследовании данных обнаруживается, что, вероятнее всего, это не проблема алгоритма, дело в самой природе этих данных – спам в рассматриваемых здесь данных нельзя назвать постоянно повторяющимся. Провайдеры электронной почты находятся в гораздо лучшем положении при использовании совместной фильтрации спама благодаря объему и разнообразию исследуемых сообщений. Если только спамер не ориентировался на атаку многих сотрудников вашей организации, в массиве образцов спама не обнаружится сколько-нибудь значительного количества повторений. Поэтому необходимо отказаться от поиска совпадений основ слова и перейти к вычислению мер сходства (коэффициентов) Жаккара, если требуется более надежное и точное решение.

В этот момент вы разочарованы результатом эксперимента и решаете выполнить дополнительные исследования, прежде чем продолжить разработку. Обнаруживается, что многие другие разработчики достигли перспективных результатов, используя методику наивной байесовской классификации (Naive Bayes classiication). После более глубокого изучения принципа работы этого алгоритма вы приступаете к созданию прототипа решения. Библиотека scikit-learn предлагает удивительно простой класс `sklearn.naive_bayes.MultinomialNB` (**https://scikit-learn.org/stable/modules/naive_bayes.html**), которым можно воспользоваться для быстрого получения результатов в текущем эксперименте. Также можно повторно использовать большой объем ранее разработанного кода для синтаксического разбора (парсинга) файлов сообщений электронной почты и для предварительной обработки меток. Но вы решаете попытаться передавать одним блоком тему

сообщения и тело сообщения как простой текст (разделенные символом перехода на новую строку) без удаления каких-либо шумовых слов и без выделения основы слова с помощью пакета NLTK. Определяется небольшая функция для чтения всех файлов электронной почты в простом текстовом формате[1]:

```
def read_e-mail_files():
    X = []
    y = []
    for i in xrange(len(labels)):
        filename = 'inmail.' + str(i+1)
        e-mail_str = extract_e-mail_text(os.path.join(DATA_DIR, filename))
        X.append(e-mail_str)
        y.append(labels[filename])
    return X, y
```

Затем используется вспомогательная функция `sklearn.model_selection.train_test_split()` (**https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.train_test_split.html**) для разделения набора данных на тренировочное и тестовое подмножества случайным образом (аргумент `random_state=123` передается для обеспечения воспроизводимости результата):

```
from sklearn.model_selection import train_test_split

X, y = read_e-mail_files()

X_train, X_test, y_train, y_test, idx_train, idx_test = \
    train_test_split(X, y, range(len(y)),
    train_size=TRAINING_SET_RATIO, random_state=2)
```

После подготовки исходных «сырых» данных необходимо выполнить некоторую дополнительную обработку токенов для преобразования каждого e-mail-сообщения в векторное представление, которое `MultinomialNB` воспринимает как входные данные.

Один из простейших способов преобразования текстового тела сообщения в вектор признаков – использование представления мешка слов (bag-of-words) с проходом по всему массиву документов и генерацией словаря токенов, используемых в этом массиве. Каждое слово в этом словаре представляет признак, и каждому признаку соответствует счетчик появлений этого слова в массиве документов. Например, рассмотрим предполагаемую ситуацию, в которой имеются только три сообщения из всего массива документов:

```
tokenized_messages: {
    'A': ['hello', 'mr', 'bear'],
    'B': ['hello', 'hello', 'gunter'],
    'C': ['goodbye', 'mr', 'gunter']
}
```

---

[1] Выбор имен переменных в нижнем регистре для одиночных столбцов значений и имен переменных в верхнем регистре для составных столбцов значений представляет собой не самое удачное соглашение для программного кода реализации машинного обучения. Этот пример можно найти в файле примечаний Python Jupyter *chapter1/spam-fighting-naivebayes.ipynb* в репозитории исходного кода для данной книги (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter1/spam-fighting-naivebayes.ipynb**).

```
# Метки столбцов вектора признаков Bag-of-words:
# ['hello', 'mr', 'doggy', 'bear', 'gunter', 'goodbye']
vectorized_messages: {
    'A': [1,1,0,1,0,0],
    'B': [2,0,0,0,1,0],
    'C': [0,1,0,0,1,1]
}
```

Этот процесс не принимает во внимание такую важную информацию, как порядок слов, структура содержимого и сходство слов, и его можно весьма просто реализовать с использованием класса `sklearn.feature_extraction.CountVectorizer` (**https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html**):

```
from sklearn.feature_extraction.text import CountVectorizer

vectorizer = CountVectorizer()
X_train_vector = vectorizer.fit_transform(X_train)
X_test_vector = vectorizer.transform(X_test)
```

Также можно попытаться использовать векторизацию статистической меры частота слова / обратная частота документа (term frequency / inverse document frequency – TF/IDF) вместо простых счетчиков. Мера TF/IDF нормализует простые счетчики слов и, вообще говоря, является более эффективным показателем статистической значимости слова в тексте. Реализация этой меры представлена как `sklearn.feature_extraction.text.TfidfVectorizer`.

Теперь можно приступить к тренировке и тестированию мультиноминального наивного байесовского классификатора:

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score

# Инициализация классификатора и прогнозирование меток
mnb = MultinomialNB()
mnb.fit(X_train_vector, y_train)
y_pred = mnb.predict(X_test_vector)

# Вывод результатов
print('Accuracy {:.3f}'.format(accuracy_score(y_test, y_pred)))
```

```
> Accuracy: 0.956
```

Точность 95,6 % на 7 % выше, чем при использовании методики LSH[1]. Это неплохой результат для нескольких строк исходного кода, который дает приблизительное представление о функциональных возможностях современных фильтров спама. Некоторые самые передовые фильтры спама в действительности управля-

[1] Вообще говоря, использование только одного критерия точности для количественной оценки эффективности модели прогнозирования – грубая и неполноценная методика. Правильная оценка модели является важной темой, которая подробно обсуждается в главе 2. Здесь мы предпочли упрощенный вариант и использовали точность как приблизительную меру эффективности. Метод `sklearn.metrics.classification_report()` предоставляет критерии `precision`, `recall`, f_1-`score` и `support` для всех классов, которые могут исследоваться в различных сочетаниях для получения более точного представления о том, насколько эффективна выбранная модель.

ются одним из вариантов наивного байесовского классификатора. В машинном обучении объединение нескольких независимых классификаторов и алгоритмов в ансамбль (ensemble) (также известный под названием стековое обобщение (stacked generalization), или стогование (stacking)) является широко распространенным способом использования очевидных преимуществ каждой методики. Таким образом, можно представить себе, до какой степени может улучшить результат сочетание черных списков слов, нечеткое сравнение хеш-значений и модель наивного байесовского классификатора.

К сожалению, выявление спама в реальной практике реализуется не так просто, как было показано в приведенном выше примере. Существует множество различных типов спама, у каждого из которых имеется собственный вектор атаки и методика уклонения от обнаружения. Например, некоторые спам-сообщения главным образом основаны на том, что читателю настойчиво предлагается щелкнуть по ссылкам. Таким образом, внутреннее содержимое сообщения может не содержать столько подозрительного текста, сколько другие типы спама. Кроме того, спам этого типа может пытаться обмануть классификаторы, выявляющие спам по ссылкам, используя более изощренные методы, такие как маскировка и цепочки перенаправления по ссылкам. Другие типы спама могут полагаться исключительно на рекламные изображения и вообще не использовать текст.

Итак, вы вполне довольны достигнутым результатом и решаете развернуть разработанное решение. Но, как всегда в этом случае, когда участниками противостояния являются люди, спамеры в конце концов обнаружат, что их сообщения не достигают цели, и могут изменить образ действий, чтобы избежать обнаружения. Поэтому необходимо постоянно усовершенствовать алгоритмы обнаружения и классификаторы и все время на шаг опережать своих противников.

В следующих главах рассматривается, как методы машинного обучения могут помочь избежать постоянного участия в бесконечной игре «прихлопни крота»[1] с атакующими злоумышленниками, а также описываются возможности создания более адаптивного решения, чтобы свести к минимуму постоянную ручную настройку и регулирование.

Ограничения машинного обучения в сфере безопасности

Представление о том, что методы машинного обучения всегда дают положительные результаты в любых случаях их практического применения, абсолютно неверно. В реальной практике обычно существуют факторы оптимизации свойств, отличающихся от релевантности, воспроизводимости и точности.

Например, объяснимость результатов классификации может быть более важной в некоторых приложениях, чем все прочие факторы. Иногда гораздо труднее выделить обоснование решения, принятого системой машинного обучения, по сравнению с простым правилом. Кроме того, некоторые системы машинного обучения могут потребовать значительно большего объема ресурсов, чем альтернативные варианты. Это часто становится главным препятствием реализации

[1] Whack-a-mole – весьма популярная аркадная игра – см. **https://en.wikipedia.org/wiki/Whac-A-Mole**. – *Прим. перев.*

методов машинного обучения в средах с ограниченными ресурсами, например во встроенных системах.

Не существует универсального алгоритма машинного обучения, который бы хорошо работал в любых предметных областях. Алгоритмы существенно отличаются по своей применимости в различных приложениях и для разнообразных наборов данных. Несмотря на то что методы машинного обучения вносят определенный вклад в общее представление об искусственном интеллекте, в настоящее время их функциональные возможности можно сравнивать с возможностями человеческого разума только в некоторых немногих аспектах.

Процесс принятия решений человеком имеет информационную поддержку в виде огромного объема контекста, основой которого служат культурная среда и экспериментальные знания. Этот процесс очень трудно воспроизвести в системах машинного обучения. Вернемся к первоначальной методике черного списка слов, который использовался в примере фильтрации спама. Когда человек исследует содержимое сообщения электронной почты, чтобы определить, спам это или не спам, процесс принятия решения никогда не бывает таким простым, как поиск существования определенных слов. Контекст, в котором используется слово из черного списка, в конечном итоге может быть обоснованно включен в содержимое сообщения, не являющегося спамом. Кроме того, спамеры могут использовать синонимы слов из черного списка в будущих сообщениях, чтобы передать тот же смысл, а упрощенный черный список не способен отреагировать на эту хитрость надлежащим образом. Система просто не обладает тем контекстом, которым обладает человек, т. е. система не знает, какой смысл заложен в конкретном слове и как его воспримет читатель. Постоянное пополнение черного списка новыми «подозрительными» словами – весьма трудоемкий процесс, который тем не менее не может обеспечить полного покрытия.

Даже если принятая модель машинного обучения превосходно работает с тренировочным набором данных, может случиться так, что на тестовом наборе она покажет плохой результат. Наиболее частая причина этого явления – в модели произошла переподгонка (overfitting) границ классификации на наборе тренировочных данных, поэтому характеристики обучения на этом наборе нельзя обобщить и распространить на другие, ранее неизвестные наборы данных. Например, применяемый фильтр спама, обученный на тренировочном наборе, в котором все сообщения содержат слова «inheritance» (наследство) и «Nigeria» (Нигерия), может сразу показать высокую эффективность на соответствующих «подозрительных» сообщениях, но ему ничего не известно о вполне легальной цепочке писем, в которой сотрудники обсуждают проблему наследства имущества в системе страхования сельскохозяйственных владений Нигерии.

С учетом всех этих ограничений необходимо применять методики машинного обучения с равными долями энтузиазма и осторожности, помня о том, что не все задачи можно сразу решить с помощью искусственного интеллекта и далеко не всегда это оптимальный способ решения.

Глава 2

Классификация и кластеризация

В этой главе рассматриваются методики машинного обучения, наиболее подходящие для задач обеспечения безопасности. После обзора некоторых основных принципов машинного обучения предлагается рабочий комплект алгоритмов машинного обучения, из которого можно выбрать максимально соответствующий конкретной поставленной задаче обеспечения безопасности. Мы попытались представить описание с подробностями, достаточными для полного понимания того, как и когда использовать тот или иной алгоритм, но при этом не ставили целью полное описание всех тонкостей и сложностей алгоритмов.

В этой главе намного больше математического материала, чем в остальных главах книги. Если вы предпочитаете пропустить математические выкладки и сразу перейти к практическому применению предлагаемых методик, то мы рекомендуем начать чтение с разделов «Машинное обучение на практике: работающий пример» и «Практические аспекты классификации», затем внимательно изучить некоторые наиболее широко применяемые алгоритмы машинного обучения с учителем и без учителя: логистическая регрессия (logistic regression), деревья решений (decision trees), леса деревьев решений (forests) и кластеризация методом *k* средних (k-means clustering).

Машинное обучение: задачи и методики

Предположим, что вы отвечаете за обеспечение компьютерной безопасности в своей компании. Вы устанавливаете межсетевые защитные экраны, поддерживаете в актуальном состоянии антифишинговые средства, обеспечиваете практическое применение безопасного кодирования и т. п. Но в конце рабочего дня все исполнительные руководители компании с беспокойством интересуются, не осталась ли без вашего внимания какая-либо уязвимость в системе защиты. Поэтому вы приходите к необходимости создания систем, способных выявлять и блокировать вредоносный трафик на любой поверхности атаки. В конечном итоге эти системы непременно должны давать четкие ответы на следующие вопросы:

- содержит ли вредоносное программное обеспечение каждый файл, передаваемый по вашей сети?
- не применяет ли кто-либо из сотрудников скомпрометированный пароль при попытке входа (регистрации) в систему?

- не содержится ли попытка фишинга в каждом принимаемом сообщении электронной почты?
- не предпринимается ли атака типа «отказ в обслуживании» (DoS) при каждом запросе к серверам в вашей сети?
- при каждом исходящем запросе, передаваемом за пределы вашей сети, не направлен ли он боту, вызывающему свой сервер для перехвата оперативного управления?

Все перечисленные выше задачи являются задачами классификации (classification), т. е. задачами принятия бинарного решения об определении сущности (природы) наблюдаемого события.

Таким образом, основной принцип вашей рабочей задачи можно обобщить следующим образом:

> выполнить классификацию всех событий в сети компании как вредоносных или допустимых (правильных).

При такой формулировке задача выглядит почти безнадежной: каким образом возможно классифицировать весь сетевой трафик? Но не следует отчаиваться. У вас есть секретное оружие: данные.

В частности, вы располагаете хронологическими журнальными записями о запусках бинарных файлов, о попытках регистрации в системе, о принятых сообщениях электронной почты, о входящих и исходящих запросах. В некоторых случаях можно даже узнать об атаках в прошлом и получить возможность логически связать эти атаки с конкретными событиями, зарегистрированными в журналах. Теперь, чтобы приступить к решению своей задачи, вы ищете в ранее зафиксированных данных шаблоны (patterns), которые являются признаками атак злоумышленников. Например, если обнаружилось, что с одного IP-адреса было выполнено более 20 запросов в секунду к вашим серверам и это продолжалось в течение 5 минут, то, вероятнее всего, производилась DoS-атака. (Возможно, в тот период ваши серверы на некоторое время утратили работоспособность под такой нагрузкой.)

После выявления шаблонов в данных следующим этапом становится кодирование этих шаблонов как алгоритма (algorithm), т. е. функции, принимающей входные данные, которые вы пытаетесь классифицировать, и выводящей бинарный (т. е. один из двух возможных) ответ: «вредоносные» (данные) или «допустимые» (данные). В нашем примере такой алгоритм будет очень простым[1]: в качестве входных данных он принимает количество запросов с некоторого IP-адреса в течение 5 минут до «подозреваемого» запроса и выводит классификатор legitimate (допустимый), если количество запросов меньше 6000, или классификатор malicious (вредоносный), если количество запросов больше 6000.

Итак, вы успешно «обучились» на определенном объеме данных и создали алгоритм для блокировки вредоносного трафика. Но некоторые детали должны вызывать беспокойство: в чем особенность числа 20? Почему бы не установить предельное значение равным 19 или 21? Или 19,77? В идеальном случае должен существовать какой-то теоретический способ определения наилучшего значения для устанавливаемого предела или число, выведенное из повседневной практики.

---

[1] Простые алгоритмы, подобные рассматриваемому здесь, обычно называют правилами (rules).

Если используется алгоритм для сканирования хронологических данных из прошлого и выявляется наилучшее правило классификации в соответствии с некоторым математическим определением «наилучшего», то такой процесс называется машинным обучением (machine learning).

В более общем смысле машинное обучение – это процесс использования хронологических («исторических») данных для создания алгоритма прогнозирования по будущим данным. Рассматриваемая выше задача является одной из задач классификации (classification): определение класса, к которому необходимо отнести новые поступающие данные (в примере – запрос). Классификация может быть бинарной (binary; двоичной), как в предыдущем примере, когда применяются только два класса, или мультиклассовой (multiclass), например, если необходимо определить, является ли вредоносное ПО программой-шантажистом, кейлоггером или троянской программой, предоставляющей удаленный доступ.

Кроме того, машинное обучение можно использовать для решения задач регрессионного анализа (regression problems), в которых производится попытка предсказания значения переменной как вещественного числа (real number). Например, необходимо спрогнозировать количество фишинговых сообщений электронной почты, получаемых сотрудниками в определенном месяце, с учетом данных об их должностях, привилегиях доступа, сроке работы в компании, рейтинге соблюдения правил безопасности и т. д. Задачи регрессионного анализа, для которых входные данные так или иначе связаны с некоторым интервалом времени, иногда называют анализом временно́го ряда, или ряда динамики (time series analysis). Например, прогноз стоимости ценной бумаги (акции) на завтра с учетом изменений стоимости в прошлом или прогноз количества зарегистрированных (активных) учетных записей из офиса в Сиэтле на основе известной хронологии. Выявление аномалий (anomaly detection) представляет собой более высокий уровень регрессионного анализа – это задача определения значительного отличия наблюдаемого значения от прогнозируемой величины, свидетельствующего о том, что происходит нечто необычное.

Машинное обучение также используется для решения задач кластеризации (clustering problems): в группе данных определяются элементы, похожие друг на друга. Например, при анализе большого объема данных интернет-трафика на некотором сайте может возникнуть необходимость в распределении запросов по группам. Некоторые кластеры могут содержать ботнеты, другие включают мобильных провайдеров, третьи объединяют обычных пользователей.

Машинное обучение может выполняться с учителем (supervised), в этом случае хронологические данные имеют метки, а для новых данных метки прогнозируются (предсказываются). Например, при наличии большого массива электронных писем, помеченных как спам и не спам, можно подготовить («натренировать») классификатор спама, который пытается определить, не является ли каждое новое входящее сообщение спамом. Другая методика – машинное обучение без учителя (unsupervised) без меток для хронологических данных. Возможно, даже ничего не известно о метках, которые вы пытаетесь предсказать, например в том случае, когда имеется неизвестное количество ботнетов, атакующих вашу сеть, при этом необходимо отличать ботнеты друг от друга. Задачи классификации и регрессионного анализа являются примерами машинного обучения с учителем, а кластеризация – это типичная форма машинного обучения без учителя.

Машинное обучение на практике: работающий пример

Как уже было отмечено ранее, машинное обучение – это процесс использования хронологических данных с целью создания алгоритма прогнозирования для новых данных, которые ранее не встречались. Рассмотрим подробно, как работает этот процесс, используя в качестве примера простой набор данных – данные транзакций при онлайновых покупках, собранные компанией розничной электронной торговли[1]. Набор данных содержит 39 221 транзакцию, в каждой из которых содержится 5 свойств или характеристик, которые можно использовать для описания транзакции, а также бинарную метку, определяющую, является ли конкретная транзакция образцом мошенничества: метка «1» обозначает мошенническую транзакцию, метка «0» присвоена корректной транзакции. Формат с разделением элементов запятой (CSV – comma-separated values) в этом наборе является стандартным способом представления данных для аналитических исследований. Первая строка в файле содержит имена для каждой позиции значений, размещенных в последующих строках. Рассмотрим подробнее, что означает каждая позиция данных на примере произвольно выбранной строки из этого файла:

```
accountAgeDays,numItems,localTime,paymentMethod,paymentMethodAgeDays,label
...
196, 1, 4.962055, creditcard, 5.10625, 0
```

Выполним преобразование в более удобную для чтения форму:

```
accountAgeDays:         196
numItems:               1
localTime:              4.962055
paymentMethod:          creditcard
paymentMethodAgeDays:   5.10625
label:                  0
```

Здесь можно видеть, что эта транзакция была выполнена пользователем, учетная запись которого создана 196 дней назад (`accountAgeDays`). Пользователь приобрел 1 экземпляр товара (`numItems`) приблизительно в 4:58 утра по своему местному времени. Оплата произведена с помощью кредитной карты (`paymentMethod`), и этот способ оплаты был добавлен приблизительно за 5 дней до транзакции (`paymentMethodAgeDays`). Установлена метка (`label`) 0, свидельствующая о том, что данная транзакция не мошенническая.

Может возникнуть вопрос: каким образом можно было бы узнать, что некоторая транзакция мошенническая. Предположим, что кто-то выполнил неавторизованную транзакцию, воспользовавшись вашей кредитной картой, но ваша бдительность помогла обнаружить этот обман. В таком случае следует подать заявление о возврате денег (chargeback) по этой транзакции, отметив, что транзакция выполнена не вами, поэтому желательно вернуть деньги вам. Похожие процессы существуют и для других способов оплаты, таких как PayPal или открытый кредит в магазине. Возврат денег – это убедительное и явное доказательство того,

[1] Этот набор данных можно найти в файле *chapter2/datasets/payment_fraud.csv* в репозитории исходного кода для данной книги (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter2/payment_fraud.csv**).

что транзакция является мошеннической. Этот факт позволяет собирать данные о мошеннических транзакциях.

Но существует причина, по которой в реальной жизни не всегда возможно воспользоваться опцией возврата денег, – продавцы получают запрос с подробным обоснованием возврата денег через много месяцев после выполнения транзакции, поскольку после отправки товара мошеннику последний исчезает бесследно. Обычно розничный продавец берет на себя компенсацию всех потерь в подобных ситуациях, из-за чего его доход может существенно уменьшиться. Финансовые потери такого рода можно смягчить, если бы существовал способ прогнозирования, т. е. определения, в какой степени транзакция похожа на мошенническую, до отправки товаров покупателю. Можно исследовать данные и вывести некоторые правила, как, например: «Если способ оплаты был добавлен в день транзакции и количество экземпляров товара не меньше 10, то транзакция мошенническая». Но при использовании такого правила может возникать слишком большое количество ложноположительных срабатываний. Как же использовать данные для нахождения наилучшего алгоритма прогнозирования? Ответ на этот вопрос дает машинное обучение.

В терминах машинного обучения каждое свойство транзакции называется признаком (feature). Наша цель – получить алгоритм машинного обучения, умеющий определять мошеннические транзакции по пяти признакам из набора данных, имеющегося в нашем распоряжении. Поскольку набор данных содержит метку, соответствующую нашей цели, т. е. выполнению прогнозирования, назовем такой массив «помеченным набором данных» (labeled dataset), на котором можно выполнять обучение с учителем. (Если бы метки не было, то наши возможности ограничивались бы только обучением с частичным привлечением учителя (semi-supervised learning) или обучением без учителя (unsupervised learning).) Идеальная система выявления мошенничества на входе принимает признаки транзакции и возвращает оценку вероятности (probability score) того, в какой степени исследуемая транзакция может считаться мошеннической. Рассмотрим подробнее создание прототипа такой системы с использованием машинного обучения.

Как и при решении задачи классификации спама в главе 1, воспользуемся удобной функциональностью библиотеки поддержки машинного обучения scikit-learn, написанной на языке Python. Кроме того, используем Pandas (**http://pandas.pydata.org/**), широко известную библиотеку анализа данных на языке Python для выполнения некоторых простых операций обработки данных. Сначала воспользуемся утилитой `pandas.read_csv()` (**http://pandas.pydata.org/pandas-docs/stable/generated/pandas.read_csv.html**) для считывания набора данных из файла в формате CSV:

```
import pandas as pd

df = pd.read_csv('ch1/payment_fraud.csv')
```

Отметим, что результат выполнения `read_csv()` сохраняется в переменной `df`, обозначенной аббревиатурой от DataFrame (**http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html**) – структуры данных библиотеки Pandas, которая представляет наборы данных в формате двумерной таблицы с поддержкой операций над строками и столбцами. Объекты DataFrame позволяют выполнять огромное количество действий с данными, но сейчас мы не будем углуб-

ляться в тонкости работы с этой структурой[1]. Воспользуемся функцией `DataFrame.sample()` для извлечения из переменной `df` фрагмента, состоящего из трех строк:

```
df.sample(3)
```

Таблица 2.1

	accountAgeDays	numItems	localTime	paymentMethod	paymentMethodAgeDays	label
31442	2000	1	4.748314	storecredit	0.000000	0
27232	1	1	4.886641	storecredit	0.000000	1
8687	878	1	4.921349	paypal	0.000000	0

Эта функция возвращает табличное представление трех случайно выбранных строк. В левом столбце находится числовой индекс каждой выбранной строки, верхняя строка содержит наименование каждого столбца. Обратите внимание на столбец нечислового типа: `paymentMethod`. В исследуемом наборе данных для этого признака допустимы только три значения: `creditcard`, `paypal` и `storecredit`. Этот признак называется качественной, или категориальной, переменной (categorical variable), так как ее значение обозначает принадлежность к определенной категории. Многие алгоритмы машинного обучения требуют, чтобы все признаки были числовыми[2]. Для преобразования категориальных переменных в числовую форму можно воспользоваться функцией `pandas.get_dummies()` (**http://pandas.pydata.org/pandas-docs/stable/generated/pandas.get_dummies.html**)[3]:

```
df = pd.get_dummies(df, columns=['paymentMethod'])
```

При внимательном изучении нового объекта DataFrame можно заметить, что в таблицу добавлены три столбца: `paymentMethod_creditcard`, `paymentMethod_paypal` и `paymentMethod_storecredit`:

```
df.sample(3)
```

Таблица 2.2

	accountAgeDays	...	paymentMethod_ creditcard	paymentMethod_ paypal	paymentMethod_ storecredit
23393	57	...	1	0	0
3355	1366	...	0	1	0
34248	19	...	1	0	0

---

[1] Подробная информация о Pandas DataFrame содержится в документации: **http://pandas.pydata.org/pandas-docs/stable/generated/pandas.DataFrame.html**.

[2] Это требование существует не для всех алгоритмов машинного обучения. Например, при использовании деревьев решений не обязательно, чтобы часть или все признаки были числовыми. Преимущество представления признаков в числовой форме заключается в том, что каждый элемент данных может быть записан как вектор в реальном векторном пространстве, и к нему применимы все методы линейной алгебры и анализа функций многих переменных при решении текущей задачи.

[3] Обычно для функции `pd.get_dummies()` мы присваиваем аргументу `drop_first` значение `True`, чтобы избежать так называемой «ловушки фиктивной переменной» (dummy variable trap), когда независимые переменные, будучи тесно связанными, искажают и даже нарушают предположения о независимости в регрессии. Мы предпочли более упрощенный вариант, чтобы избежать путаницы, но эта проблема подробно обсуждается в главе 5.

Каждый новый признак является бинарным (т. е. принимает только значение 0 или 1), и в каждой строке только для одного из этих признаков установлено значение 1. Такая методика определения (кодирования) категориальной переменной называется унитарным кодированием (one-hot encoding; unitary encoding). Определяемые по этой методике переменные называют фиктивными переменными (dummy variables) в терминологии математической статистики.

Теперь можно разделить общий массив данных на тренировочный и тестовый наборы (как это было сделано в главе 1).

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(
    df.drop('label', axis=1), df['label'],
    test_size=0.33, random_state=17)
```

Функция `sklearn.model_selection.train_test_split()` позволяет разделить исследуемый массив данных на тренировочный и тестовый наборы. Отметим, что в первом аргументе функции передается значение `df.drop('label', axis=1)`. Разделение на наборы `X_train` и `X_test` выполняется в соотношении 0.67:0.33, поскольку передан аргумент `test-size=0.33`, определяющий, что две трети массива данных необходимо использовать для тренировки алгоритма машинного обучения, а оставшуюся треть, т. е. тестовый набор, – для проверки правильности работы алгоритма. Столбец `label` исключается из `X` перед разделением на `X_train` и `X_test` и передается в функцию как подмножество `y-df['label']`. После этого метки также будут разделены в том же соотношении на наборы `y_train` и `y_test`.

Применим стандартный алгоритм обучения с учителем – логистическую регрессию (logistic regression) – к нашим данным:

```
from sklearn.linear_model import LogisticRegression
clf = LogisticRegression()
clf.fit(X_train, y_train)
```

В первой строке импортируется класс `sklearn.linear_model.LogisticRegression` (**https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html**). Затем во второй строке инициализируется объект `LogisticRegression` с помощью вызова конструктора класса. В третьей строке `X_train` и `y_train` (т. е. тренировочный набор) передаются в функцию `fit()`, результатом работы которой является тренировочная модель классификатора, сохраняемая в объекте `clf`. Этот классификатор принимает тренировочные данные и применяет логистическую регрессию (эту методику мы более подробно рассмотрим в следующем разделе) для выделения некоторых обобщающих характеристик мошеннических и корректных транзакций в формируемую модель.

Для прогнозирования с использованием этой модели теперь необходимо просто передать непомеченные признаки в функцию `predict()` созданного классификатора:

```
y_pred = clf.predict(X_test)
```

Проверяя `y_pred`, можно увидеть, что прогнозируемые метки присвоены каждой строке в тестовом наборе `X_test`. Отметим, что во время тренировки классификатор не имел доступа к набору меток `y_test`, поэтому прогнозы в `y_pred` являются

исключительно результатами обобщений, выведенных при работе с тренировочным набором. Затем применяется функция `sklearn.metrics.accuracy_score()` (которая использовалась и в главе 1) для получения оценки точности выполненных прогнозов:

```
from sklearn.metrics import accuracy_score

print(accuracy_score(y_pred, y_test))

> 0.99992273816
```

Это очень высокая точность. Но в главе 1 уже было отмечено, что оценка точности часто становится вводящим в заблуждение чрезмерным упрощением и далеко не самой лучшей метрикой для оценки подобных результатов. Вместо нее следует сгенерировать матрицу несоответствий:

```
from sklearn.metrics import confusion_matrix
print(confusion_matrix(y_test, y_pred))
```

Таблица 2.3

0	Прогнозируемые немошеннические	Прогнозируемые мошеннические
Действительно немошеннические	12 753	0
Действительно мошеннические	1	189

Здесь обнаружен только один случай неправильной классификации во всем тестовом наборе. 189 транзакций верно помечены как мошеннические. В одном случае произошло ложноотрицательное срабатывание, т. е. мошенническая транзакция не была выявлена. Ложноположительных срабатываний нет.

Ниже приведен исходный код, использованный для тренировки и тестирования модели выявления мошеннических платежных транзакций на основе логистической регрессии[1]:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score, confusion_matrix

# Считывание данных из CSV-файла
df = pd.read_csv('ch1/payment_fraud.csv')

# Преобразование категориального признака в фиктивные переменные с унитарным кодированием
df = pd.get_dummies(df, columns=['paymentMethod'])

# Разделение массива данных на тренировочный и тестовый наборы
X_train, X_test, y_train, y_test = train_test_split(
    df.drop('label', axis=1), df['label'],
    test_size=0.33, random_state=17)

# Инициализация и тренировка модели классификатора
clf = LogisticRegression().fit(X_train, y_train)
```

[1] Этот пример можно найти в файле примечаний Python Jupyter *chapter2/logisic-regression-fraud-detection.ipynb* в репозитории исходного кода для данной книги.

```
# Выполнение прогнозирования на тестовом наборе
y_pred = clf.predict(X_test)

# Сравнение прогнозов по тестовому набору с действительными метками
print(accuracy_score(y_pred, y_test))
print(confusion_matrix(y_test, y_pred))
```

Эту модель можно применять к любой входящей транзакции и получать оценку вероятности того, что данная транзакция является мошеннической (или немошеннической):

```
clf.predict_proba(df_real)

  # Массив, представляющий вероятность того, что исследуемая транзакция
  # имеет метку 0 (в позиции 0) или 1 (в позиции 1)
> [[  9.99999994e-01   5.87025707e-09]]
```

Принимая `df_real` как DataFrame с одной строкой, представляющей входящую транзакцию, принятую онлайн-продавцом, классификатор прогнозирует, что эта транзакция с вероятностью 99,9 999 994 % не является мошеннической (напомним, что `y=0` означает «немошенническая»).

Возможно, вы заметили, что вся работа по машинному обучению, т. е. та часть, где происходило обучение алгоритма прогнозирования, была «спрятана» в единственном API-вызове библиотеки scikit-learn `LogisticRegression.fit()`. Но что в действительности происходило в этом черном ящике, который позволил обучить модель правильному прогнозированию мошеннических транзакций? Откроем черный ящик и внимательно рассмотрим его содержимое.

Тренировка алгоритмов машинного обучения

Сущность алгоритма машинного обучения заключается в приеме на входе тренировочного набора данных (training dataset) и выдаче на выходе модели (model). Модель – это алгоритм, принимающий новые элементы (порции) данных в той же форме, что и тренировочные данные, и выдающий итоговый прогноз. Все алгоритмы машинного обучения определяются по трем взаимозависимым компонентам:

- семейство моделей (model family), определяющее пространство моделей, из которого можно выбрать наиболее подходящую;
- функция потерь (loss function), позволяющая сравнивать различные модели в числовом выражении;
- процедура оптимизации (optimization procedure), позволяющая выбрать наилучшую модель в семействе.

Рассмотрим каждый компонент более подробно.

Семейства моделей

Вспомним, что мы формировали набор данных для определения мошеннических транзакций из семи числовых признаков: четыре признака были взяты непосредственно из исходных данных и три признака были созданы с помощью унитарного кодирования способа оплаты. Таким образом, можно представить каждую транзакцию как точку в семимерном реальном векторном пространстве. Наша

цель – разделить это пространство на области мошеннических и немошеннических транзакций. «Модель», полученная как результат работы выбранного алгоритма машинного обучения, представляет собой описание такого разделения векторного пространства.

Теоретически разделение векторного пространства на области мошеннических и немошеннических транзакций может быть бесконечно сложным. На практике большинство алгоритмов создает границу решения (decision boundary) (или поверхность решения – decision surface), представляющую собой некоторую поверхность в исследуемом векторном пространстве[1]. С одной стороны от границы решения находятся точки, помеченные как мошеннические, на другой стороне – немошеннические точки. Граница может быть простой линией (или гиперплоскостью в измерениях с более высокой размерностью) или сложной комбинацией из нескольких кривых линий, ограничивающих не связанные между собой области. На рис. 2.1 показано несколько примеров границ решения.

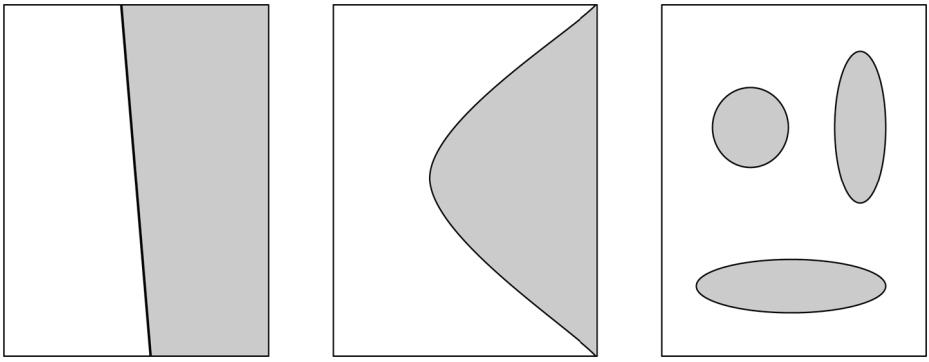


Рис. 2.1 ❖ Примеры двумерных пространств, разделенных различными типами границ решения

Если необходима более точная детализация, то вместо установления соответствия между каждой точкой векторного пространства и одним из возможных значений «мошенническая» и «немошенническая» можно определить связь каждой точки с вероятностью (probability) того, является ли она мошеннической. В этом случае выбранный алгоритм машинного обучения выдаст функцию, которая присваивает каждой точке векторного пространства значение от 0 до 1. В нашем примере эти значения интерпретируются как вероятность того, является ли точка (транзакция) мошеннической.

В любом алгоритме машинного обучения существуют внутренние ограничения при определении конкретного типа границы решений или функции вычисления вероятности. Эти ограничения могут быть описаны конечным числом параметров модели (model parameters). Самой простой является линейная граница решения (linear decision boundary), т. е. гиперплоскость в векторном пространстве. Ориентированная гиперплоскость H в n-мерном векторном пространстве может

[1] С технической точки зрения это дифференцируемая (т. е. непрерывная) ориентированная поверхность.

быть описана n-мерным вектором θ, ортогональным к этой гиперплоскости, и дополнительным вектором β, определяющим, насколько далеко эта гиперплоскость удалена от начала координат:

$$H: \theta \cdot (x - \beta) = 0.$$

Это описание позволяет разделить векторное пространство на две части. Для присваивания вероятностей необходимо определить расстояние точки x от гиперплоскости H. Следовательно, можно вычислить «оценку» как действительное число:

$$s(x) = \theta \cdot (x - \beta) = \theta \cdot x + b,$$

где $b = -\theta \cdot \beta$. Таким образом, наша модель вычисления оценки может быть описана n + 1 параметрами модели: n параметров для описания вектора θ и один «компенсирующий» параметр b. Для преобразования оценки в классификацию просто выбирается пороговое значение t. Все оценки, превышающие t, обозначают мошеннические точки, а оценки ниже t обозначают немошеннические точки.

Если требуется установить связь действительного числового значения оценки s(x) с вероятностью, то необходимо применить функцию, отображающую действительные числа в значения из интервала [0, 1]. Стандартная функция для этой цели известна под названием логистическая, или сигмоидная, функция (logistic/sigmoid function)[1], которая показана на рис. 2.2. Функция определяется следующей формулой:

$$h_\theta(x) = \frac{1}{1 + e^{-\theta^T x}}.$$

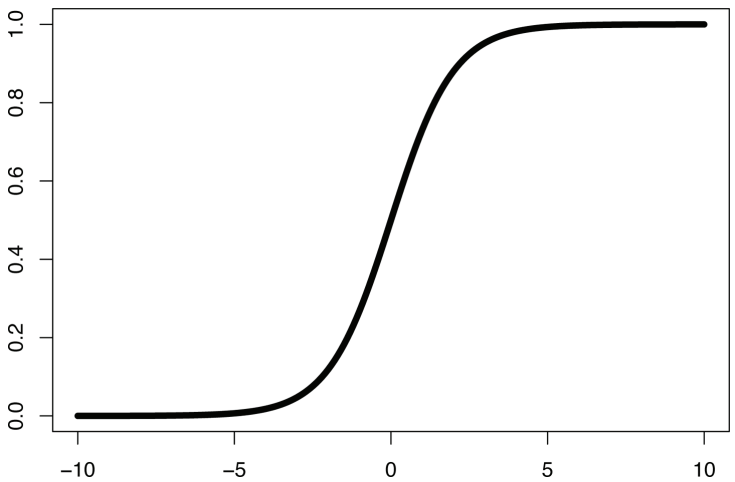


Рис. 2.2 ❖ Сигмоидная кривая (функция)

[1] Полное обоснование причин, по которым именно эта функция выбрана в качестве гипотетической функции для бинарной логистической регрессии, занимает слишком много места, поэтому здесь мы не будем углубляться в детали. Подробное описание см. в разделе 4.4 книги The Elements of Statistical Learning, 2nd ed., by Trevor Hastie, Robert Tibshirani and Jerome H. Friedman (изд. Springer).

Данные, выводимые логистической функцией, могут интерпретироваться как значения вероятности, что позволяет определить правдоподобие зависимого значения с привлечением к вычислениям конкретного значения из входного вектора признаков x.

Функция потерь

После установления ограничений на выбор алгоритмов прогнозирования, определяющих конкретное параметризованное семейство, необходимо выбрать наилучший алгоритм для предложенного тренировочного набора данных. Но как узнать, что выбран действительно самый лучший алгоритм? Наилучший алгоритм должен оптимизировать некоторую числовую характеристику, вычисленную по исследуемым данным. Эта числовая характеристика называется целевой функцией (objective function). При использовании машинного обучения целевую функцию называют также функцией стоимости (cost function) или функцией потерь (loss function), поскольку она позволяет оценить количественно «стоимость» неправильных прогнозов или «потери», связанные с ними.

С математической точки зрения функция потерь – это функция, отображающая набор пар значений (прогнозируемая метка (predicted label), истинная метка (truth label)) в действительное число. Цель алгоритма машинного обучения – найти такие параметры модели, при которых прогнозируемые метки, получаемые при обработке тренировочного набора данных, минимизируют функцию потерь.

В задачах регрессии, где для каждого алгоритма прогнозирования выводится действительное число вместо метки, стандартная функция потерь представляет собой сумму квадратичных ошибок (sum of squared errors). Если y_i – истинное значение, а $\hat{y}_i$ – прогнозируемое значение, то функция потерь выглядит следующим образом:

$$C(Y) = \sum_i (\hat{y}_i - y_i)^2.$$

Эту функцию потерь можно также использовать для задач классификации, где y_i принимает одно из двух значений, 0 или 1, а $\hat{y}_i$ является вероятностной оценкой, получаемой по результатам работы алгоритма.

Для задач логистической регрессии в качестве функции потерь используется функция отрицательного логарифмического правдоподобия (negative log likelihood). Правдоподобие (likelihood) набора вероятностных прогнозов $\{p_i\}$ для заданного набора достоверных меток $\{y_i\}$ определяется как вероятность, с которой эти истинные метки должны обнаруживаться при выборке из набора биномиальных распределений, соответствующих вероятностям $\{p_i\}$. В частности, если истинная метка y_i равна 0, а правдоподобие значения 0 равно $1 - p_i$, т. е. вероятности того, что значение 0 должно появляться в выборке биномиального распределения с меаной p_i. Если истинная метка y_i равна 1, то правдоподобие равно p_i.

Правдоподобие всего набора прогнозов представляет собой произведение отдельных правдоподобий:

$$\mathscr{L}(\{p_i\}, \{y_i\}) = \prod_{y_i=0} (1 - p_i) \cdot \prod_{y_i=1} p_i.$$

Целью логистической регрессии является нахождение параметров, при которых вычисляются вероятности $\{p_i\}$, максимизирующие правдоподобие.

Для упрощения вычислений, в особенности потому, что большинство методов оптимизации требует вычисления производных функции потерь, мы будем применять функцию отрицательного логарифмического правдоподобия. Поскольку максимизация правдоподобия равнозначна минимизации отрицательного логарифмического правдоподобия, можно считать функцию отрицательного логарифмического правдоподобия функцией потерь:

$$\ell(\{p_i\}, \{y_i\}) = -\sum_i ((1-y_i)\log(1-p_i) + y_i \log p_i).$$

Здесь использован тот факт, что значение y_i всегда равно только 0 или 1, для объединения двух произведений в одно выражение.

Оптимизация

Последним этапом в процедуре машинного обучения является поиск оптимального набора параметров, которые минимизируют функцию потерь. Для реализации процесса поиска используется алгоритм оптимизации (optimization algorithm). Существует множество разнообразных алгоритмов оптимизации, из которого вы можете выбрать наиболее соответствующий вашей модели машинного обучения[1]. Большинство оценочных функций библиотеки scikit-learn (например, `LogisticRegression`) позволяет определить числовую решающую функцию (numerical solver) для практического применения, но каковы различия между предлагаемыми вариантами и как выбрать наиболее подходящий?

Задача алгоритма оптимизации – минимизировать (или максимизировать) целевую функцию. В машинном обучении целевая функция выражается с помощью параметров модели обучения (θ и b в предыдущем примере), а ее целью является поиск таких значений θ и b, которые оптимизируют целевую функцию.

Алгоритмы опимизации можно разделить на две основные группы:

- алгоритмы первого порядка (first-order algorithms): эти алгоритмы оптимизируют целевую функцию, используя первую производную этой функции с учетом параметров обучения. Методы градиентного спуска (gradient descent) представляют наиболее широко используемый тип алгоритмов оптимизации первого порядка. Их можно применять для поиска входных данных функции, которая выдает минимальное (или максимальное) значение. Вычисление градиента функции (т. е. частных производных по каждой переменной) позволяет определить моментальное направление, в котором должны смещаться значения параметров, чтобы достичь наиболее оптимального результата, выдываемого функцией;
- алгоритмы второго порядка (second-order algorithms): из названия понятно, что эти алгоритмы используют вторые производные для оптимизации

[1] Основной книгой, в которой определяется область конвексной оптимизации (convex optimization) (следует отметить, что не все обсуждаемые здесь задачи оптимизации могут быть конвексными, т. е. выпуклыми (convex) по своей сущности), является Convex Optimization by Stephen P. Boyd and Lieven Vandenberghe (издательство Cambridge University Press).

целевой функции. Алгоритмы второго порядка лишены такого недостатка, как медленная сходимость. Например, алгоритмы второго порядка хорошо работают при решении задачи нахождения точек седловины (перевала кривой), тогда как алгоритмы первого порядка, вероятнее всего, будут поставлены в тупик на этих точках. Тем не менее методы второго порядка чаще всего являются более медленными и требуют больших издержек на вычисления.

Методы первого порядка используются чаще по причине их относительной эффективности. Выбор подходящего алгоритма оптимизации зависит от размера набора данных, от сущности функции стоимости, от типа задачи обучения и от скорости/требований к ресурсам для выполнения предполагаемой операции. Кроме того, некоторые методики регуляризации также могут создавать проблемы совместимости с определенными типами методов оптимизации. К группе алгоритмов первого порядка относятся следующие:

- LIBLINEAR[1] – принятый по умолчанию метод решения для линейной оценки в библиотеке scikit-learn. Этот алгоритм не очень эффективен для больших наборов данных, поэтому в документации на библиотеку рекомендуется использовать методы Stochastic Average Gradient (SAG) или SAGA (усовершенствованный метод SAG), работающие лучше с большими наборами данных[2]. Различные алгоритмы оптимизации по-разному работают в задачах мультиклассовой классификации. LIBLINEAR работает только с бинарной классификацией. Чтобы применить его в случае мультиклассовой классификации, необходимо воспользоваться схемой «один против остальных» (one-versus-rest), которая будет рассмотрена более подробно в главе 5;
- Stochastic Gradient Descent (SGD) – очень простой и эффективный алгоритм оптимизации, который выполняет обновление параметра для каждого отдельного тренировочного экземпляра данных. Стохастическая (случайная) сущность градиентного спуска означает, что этот алгоритм с большей вероятностью находит новые и, возможно, лучшие локальные минимумы по сравнению со стандартным методом градиентного спуска. Но обычно он дает результаты с высокодисперсными отклонениями, которые могут привести к более медленной сходимости. Эту проблему можно устранить с помощью снижения скорости обучения (например, с экспоненциальным снижением скорости обучения), что позволяет уменьшить флуктуации и достичь более быстрой сходимости алгоритма.
  Методика с названием моментум (momentum) также помогает ускорить сходимость метода SGD посредством направления оптимизации только в необходимых направлениях и сведения к минимуму любых перемещений в ненадлежащих направлениях. Такой подход позволяет стабилизировать работу метода SGD;
- алгоритмы оптимизации, такие как AdaGrad, AdaDelta и Adam (Adaptive Moment Estimation), позволяют разделять и адаптировать скорости обучения для каждого параметра и решать некоторые задачи с помощью других более простых алгоритмов градиентного спуска;

---

[1] *Rong-En Fan et al.* LIBLINEAR: A Library for Large Linear Classification. Journal of Machine Learning Research 9 (2008): 1871–1874.

[2] *Francis Bach*. Stochastic Optimization: Beyond Stochastic Gradients and Convexity. INRIA – Ecole Normale Supérieure, Paris, France. Joint tutorial with Suvrit Sra, MIT – NIPS, 2016.

- если тренировочный набор данных слишком велик, то потребуется применение алгоритма распределенной оптимизации. Одним из таких широко известных алгоритмов является метод переменных направлений множителей Alternating Direction Method of Multipliers (ADMM)[1].

Пример: метод градиентного спуска

В конце раздела мы кратко рассмотрим подробности практического использования метода градиентного спуска, мощного алгоритма оптимизации, который можно применять в разнообразных задачах машинного обучения.

Стандартный алгоритм градиентного спуска можно описать в виде следующих шагов.

1. Выбор случайных начальных параметров для модели машинного обучения. В варианте с линейной моделью это означает выбор случайного нормального вектора θ и смещения β, которые в результате дают случайную гиперплоскость в n-мерном пространстве.
2. Вычисление значения градиента функции потерь для этой модели в точке, описываемой выбранными параметрами.
3. Изменение параметров модели в направлении максимального уменьшения градиента по определенному малому модулю (абсолютной величине), обычно обозначаемой как α, или скорость обучения.
4. Итерация: повторение шагов 2 и 3 до тех пор, пока не будет выполнено условие сходимости или не будет достигнут удовлетворительный уровень оптимизации.

На рис. 2.3 показаны промежуточные результаты процесса оптимизации метода градиентного спуска в задаче линейной регрессии. При нуле итераций видно, что линия регрессии, сформированная с помощью случайно выбранных параметров, абсолютно не соответствует набору данных. Можно понять, что в этот момент значение суммы квадратов как функции стоимости слишком велико. При трех итерациях заметно, что линия регрессии очень быстро перемещается в более разумное положение. Между 5 и 20 итерациями линия регрессии медленно подводится в постепенно улучшающиеся оптимальные положения, где функция стоимости минимальна. Если дополнительно выполняемые итерации не дают существенного уменьшения значения функции стоимости, то можно считать, что удалось достичь сходимости процесса оптимизации и получены окончательные параметры обучения для тренировочной модели.

Как выбрать алгоритм оптимизации

Как и во многих случаях в науке о данных, универсального алгоритма оптимизации «на все случаи жизни» не существует. Кроме того, нет ясных и четких правил, по которым алгоритм может быть признан абсолютно лучшим для выполнения конкретных типов задач. Чаще всего необходимо некоторое количество проб и ошибок, для того чтобы найти алгоритм, соответствующий требованиям и позволяющий решить поставленную задачу. Кроме сходимости и скорости, существует множество других критериев, которые необходимо учитывать при выборе алгоритма оптимизации. В общем случае приемлемая стратегия заключается

---

[1] *Stephen Boyd et al*. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. Foundations and Trends in Machine Learning 3 (2011): 1–122.

в том, чтобы начать с варианта, предлагаемого по умолчанию, или с варианта, выглядящего наиболее разумным, и постепенно применять все возможные усовершенствования.

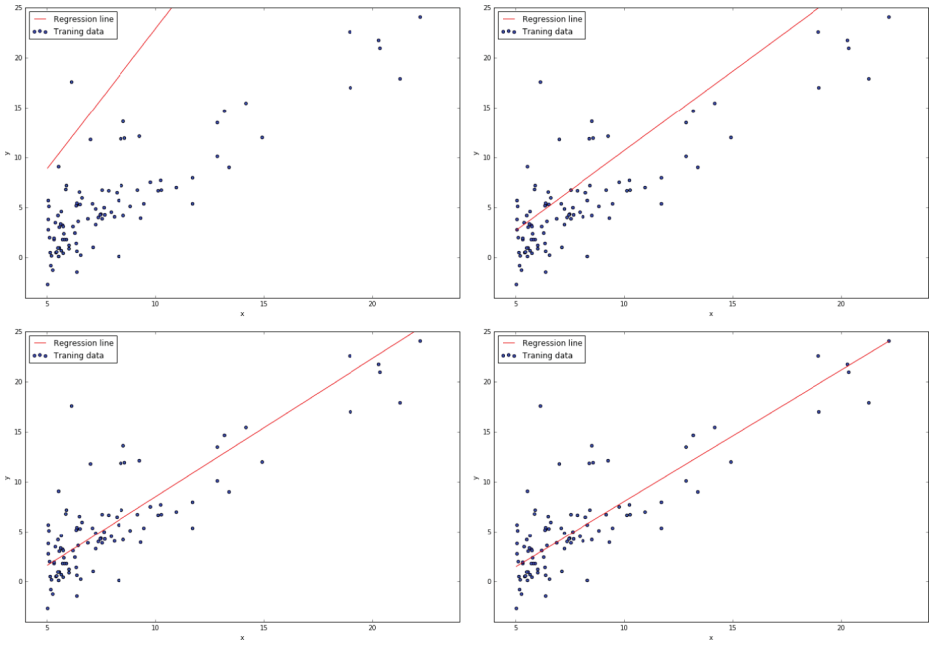


Рис. 2.3 ❖ Линия регрессии после увеличивающегося числа итераций при оптимизации методом градиентного спуска (состояния после 0, 3, 5 и 20 итераций градиентного спуска показаны на верхнем левом, верхнем правом, нижнем левом и нижнем правом графиках соответственно)

Алгоритмы классификации с учителем

После ознакомления с основными принципами работы алгоритмов машинного обучения перейдем к кратким описаниям некоторых наиболее часто применяемых алгоритмов обучения с учителем для задач классификации.

Логистическая регрессия

Некоторые аспекты логистической регрессии уже обсуждались выше, но сейчас мы более подробно рассмотрим главные свойства этого алгоритма. Логистическая регрессия принимает в качестве входных данных векторы числовых признаков и пытается спрогнозировать логарифм отношения шансов (log odds)[1] появления каждого элемента (точки) данных. Можно преобразовать логарифм отношения шансов в значения вероятности с помощью сигмоидной функции, описанной выше. В пространстве логарифмов отношения шансов граница решения является прямой линией, поэтому увеличение значения признака монотонно увеличивает

[1] Для события X, возникающего с вероятностью p, шансы (odds) возникновения X равны p/(1 – p), а логарифм отношения шансов имеет вид log (p/(1 – p)).

или уменьшает (в зависимости от знака коэффициента) оценку результата, выводимого моделью.

Почему не линейная регрессия?

Линейная регрессия, изучаемая в любом начальном курсе математической статистики, является мощным инструментом прогнозирования будущих результатов на основе прошлых данных. Алгоритм принимает данные, состоящие из исходных переменных (представленных как векторы в некотором векторном пространстве) и переменной ответа (действительное число), и вычисляет «наиболее соответствующую» линейную модель, которая отображает каждый элемент (точку) в векторном пространстве в прогнозируемый ответ. Почему бы не воспользоваться этим методом для решения задач классификации? Проблема в том, что линейная регрессия прогнозирует действительные значения переменной, а при классификации необходимо прогнозировать категориальную переменную. Если попытаться отобразить две категории в значения 0 и 1 и выполнить линейную регрессию, то в итоге получится линия, устанавливающая соответствие между исходными переменными и некоторым выходным результатом, как показано на рис. 2.4. Но что означает этот результат? Невозможно интерпретировать его как вероятность, потому что принимаемые на входе значения могут быть меньше 0 и больше 1, как видно по рис. 2.4. Можно было бы интерпретировать результат как оценку и выбрать пороговое значение для границы определенного класса, но если с чисто технической точки зрения такой подход сработает, то он все равно не позволит провести правильную классификацию. Причина в том, что квадратичная функция потерь, используемая в линейной регрессии, не может точно отобразить, насколько далеко точки находятся от границы классификации: в примере на рис. 2.4 точке X = 1 соответствует бóльшая ошибка, чем точкам со значениями X = 10 и больше, даже несмотря на то что эти точки расположены дальше от границы классификации (например, точка X = 50).

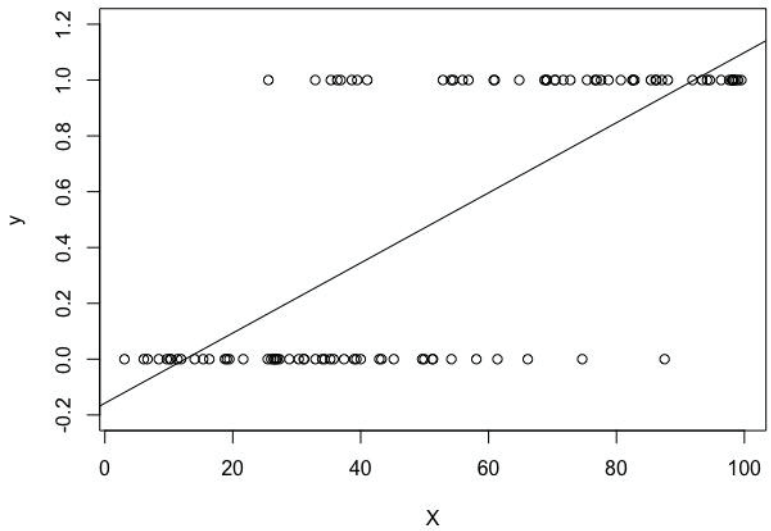


Рис. 2.4 ❖ Использование линейной регрессии для классификации

Логистическая регрессия является одним из наиболее часто применяемых на практике алгоритмов благодаря ряду полезных свойств: возможность весьма эффективной тренировки, в том числе и в распределенном варианте, способность успешно работать с миллионами признаков, допустимость краткого описания и быстрой оценки алгоритма (простое скалярное произведение), наконец, объяснимость – может быть вычислен вклад каждого признака в итоговую оценку.

Но при этом существует несколько важных фактов, которые необходимо знать при рассмотрении логистической регрессии как методики обучения с учителем:

- логистическая регрессия подразумевает линейность признаков (независимых переменных) и логарифмов отношения шансов, т. е. требует, чтобы между признаками и логарифмами отношения шансов существовала линейная зависимость. Если это предположение не соблюдается, то модель будет работать некорректно;
- признаки должны иметь небольшую или полностью отсутствующую мультиколлинеарность[1], т. е. независимые переменные действительно должны быть независимыми друг от друга;
- для логистической регрессии обычно требуется более крупный размер выборки данных по сравнению с другими алгоритмами машинного обучения, такими как линейная регрессия. Числовые оценки максимального правдоподобия (используемые в логистической регрессии) менее значимы, чем обычный метод наименьших квадратов (используемый в линейной регрессии), который по требованию генерирует больше тренировочных выборок для достижения аналогичной мощности статистического обучения[2].

Деревья решений

Деревья решений представляют собой весьма универсальные и гибкие модели обучения с учителем, которые обладают важным свойством – простотой интерпретации. Из названия понятно, что дерево решений – это структура данных в виде бинарного дерева, используемая для принятия решений. Деревья представляют собой вполне интуитивно понятный способ визуального представления и анализа данных, поэтому весьма широко используются даже вне пределов области машинного обучения. Эти структуры предоставляют возможность прогнозирования как категориальных значений (деревья классификации), так и значения в формате действительных чисел (деревья регрессии), а также способны содержать и числовые, и категориальные данные без каких-либо операций нормализации или создания фиктивных переменных[3]. Благодаря этим свойствам легко

---

[1] Это предположение относится не только к логистической регрессии. Большинство других алгоритмов машинного обучения также требует отсутствия корреляции между признаками.

[2] Для выполнения логистической регрессии на небольших наборах данных рекомендуется рассмотреть применение точной логистической регрессии (exact logistic regression).

[3] Отметим, что в конце 2017 года реализация деревьев решения в библиотеке scikit-learn (`sklearn.tree.DecisionTreeClassifier` и другие методы обучения на основе деревьев) некорректно обрабатывала категориальные данные. Категориальные переменные, кодируемые с помощью целочисленных меток (например, `sklearn.preprocessing.LabelEncoder` и not `sklearn.preprocessing.OneHotEncoder` или функции `pandas.get_dummies()`), будут неправильно интерпретированы как числовые переменные. Несмотря на то что служба под-

понять, почему применение деревьев решений широко распространено в машинном обучении.

Рассмотрим более подробно процедуру создания обычного обучающего дерева решений (структуры, формируемой сверху вниз).

1. Начиная с корня (root) дерева весь набор данных разделяется на два подмножества-потомка на основе бинарного условия. Например, если определено условие «возраст ⩾ 18», то все элементы данных, для которых это условие истинно, направляются в левый набор-потомок, а элементы, для которых условие ложно, – в правый набор-потомок.
2. Далее подмножества-потомки продолжают рекурсивно разделяться на более мелкие подмножества на основе других условий. Условия разделения автоматически выбираются на каждом шаге в зависимости от того, какое условие наилучшим образом разделяет текущий набор элементов. Существует несколько общепринятых метрик, по которым количественно оценивается качество разделения:
   – мера неоднородности Джини (Gini impurity): если элементы в подмножестве были произвольным образом помечены в соответствии с распределением меток в наборе, то относительная доля неправильно помеченных элементов должна быть определена как мера неоднородности Джини. Например, если в подмножестве 25 % элементов имеют метку 0 (соответственно 75 % элементов с меткой 1), то присваивание метки 0 случайно выбранным 25 % от всех элементов (остальным присваивается метка 1) должно давать 37,5 % неправильных меток: 75 % элементов с меткой 0 и 25 % элементов с меткой 1 должны быть некорректными. Операция разделения дерева решений более высокого качества должна разделять набор данных на подмножества, четко определяемые по их меткам, следовательно, в результате мера неоднородности Джини получается более низкой. Таким образом, коэффициент неправильной классификации должен быть низким, если большинство элементов в наборе принадлежит к одному и тому же классу;
   – уменьшение дисперсии (variance reduction) часто используется в деревьях регрессии с постоянно присутствующей (распространяющейся по дереву) зависимой переменной. Уменьшение дисперсии определяется как суммарное уменьшение дисперсии в наборе, разделенном на два подмножества-потомка. Наилучшим вариантом разделения в узле дерева решений должно считаться разделение, результатом которого является наибольшее снижение дисперсии;

держки и сопровождения библиотеки scikit-learn заявляет (**https://github.com/scikit-learn/scikit-learn/issues/5442**), что модель sklearn.tree.RandomForestClassifier и прочие аналогичные модели должны работать «весьма надежно и устойчиво с категориальными признаками, некорректно кодируемыми как целочисленные признаки в практических приложениях», в силе остается настоятельная рекомендация: необходимо выполнять преобразование категориальных переменных в фиктивные или унитарные переменные перед вводом их в деревья решений sklearn. В 2018 году должна была появиться новая функциональная возможность (**https://github.com/scikit-learn/scikit-learn/pull/4899**) для методов обучения на основе деревьев, которая обеспечивает поддержку правильного разделения по категориям (до 64 категорий для каждого признака).

   - прирост информации (information gain) – это мера однородности (purity) подмножеств, полученных в результате разделения. Прирост информации вычисляется с помощью вычитания взвешенной суммы энтропии каждого узла-потомка дерева решений из энтропии родительского узла. Чем меньше энтропия потомков, тем больше прирост информации, следовательно, более качественно выполнено разделение.
3. Существует несколько различных методов для определения момента остановки процедуры разделения узлов:
   - когда все листья дерева являются однородными (pure), т. е. каждый лист-узел содержит только элементы, принадлежащие одному и тому же классу, следовательно, разделение прекращается;
   - когда очередная ветвь дерева достигает некоторой предварительно определенной максимальной глубины (maximum depth), дальнейшее разделение ветвей прекращается;
   - когда один (любой) из узлов-потомков содержит количество элементов, меньшее, чем минимальное количество элементов выборки (minimum number of samples), такой узел больше не подразделяется.
4. В завершение процесса алгоритм выводит структуру дерева, в которой каждый узел представляет бинарное решение, потомки каждого узла представляют два возможных выходных результата этого решения, а каждый лист представляет классификацию элементов данных в соответствии с путем от корня дерева до этого листа. (Для неоднородных (impure) листьев решение определяется большинством голосов в тренировочной выборке данных в соответствующем листе.)

Важным свойством деревьев решений является относительная простота объяснения классификации или результатов регрессии, так как каждый прогноз может быть выражен в виде последовательности логических условий, позволяющей проследить путь от корня дерева до любого узла-листа. Например, если модель дерева решений предсказывает, что рассматриваемый образец вредоносного ПО принадлежит к семейству A вредоносных программ, то нам точно известны причины: бинарный файл подписан до 2015 года, он не связан с конкретно определенным окном в управляющей рабочей среде, выполняет многочисленные исходящие сетевые вызовы, направленные на IP-адреса, находящиеся на территории России, и т. д. Поскольку каждому образцу выборки соответствует полная высота бинарного дерева в предельном случае (временна́я сложность O(log n)), деревья решений также эффективны для тренировки и последующего прогнозирования. Таким образом, деревья решений можно считать наиболее предпочтительным вариантом для больших наборов данных.

Тем не менее деревьям решений присущи некоторые ограничения:

- в деревьях решений часто возникает проблема переподгонки (overfitting), когда деревья становятся чрезмерно сложными и не способны правильно обобщить тренировочный набор данных. Для уменьшения сложности деревьев в качестве регулирующей методики применяется отсечение (pruning) (малоперспективных ветвей);
- деревья решений менее эффективны в случаях представления некоторых типов отношений, по сравнению с другими типами. Например, на рис. 2.5 и 2.6 показано минимальное дерево решений, требуемое для представле-

ния отношений AND, OR и XOR. Обратите внимание на то, что для XOR требуется один дополнительный промежуточный узел и дополнительная процедура разделения для правильного представления этой операции даже в таком простом примере. В реальных наборах данных это может приводить к чрезвычайно быстрому росту сложности модели;

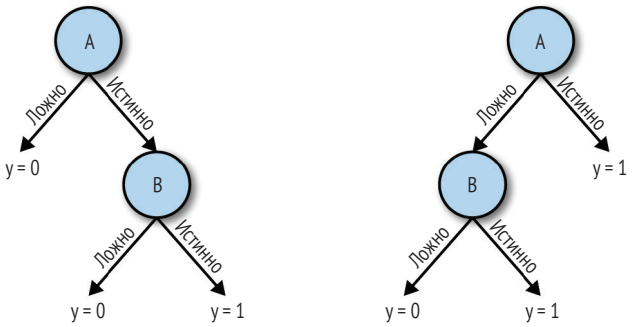


Рис. 2.5 ❖ Дерево решений для отношения A-AND-B → y = 1 (слева) и отношения A-OR-B → y = 1 (справа)

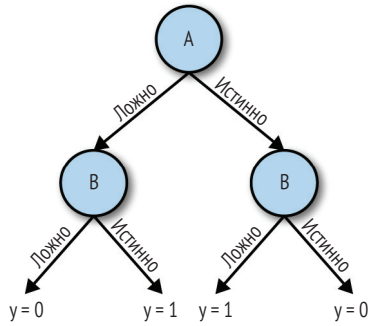


Рис. 2.6 ❖ Дерево решений для отношения A-XOR-B → y = 1

- деревья решений иногда менее точны и надежны (устойчивы), чем другие методики обучения с учителем. Незначительные изменения в тренировочном наборе данных могут приводить к весьма существенным изменениям в дереве, что, в свою очередь, влечет за собой изменения в модели прогнозов. Это означает, что деревья решений (и большинство других подобных моделей) неприменимы для онлайнового обучения или инкрементального обучения;
- метрики качества разделения для категориальных переменных в деревьях решений имеют тенденцию смещаться в сторону переменных с более вероятными значениями, т. е. разделение по непрерывным переменным или по категориальным переменным с тремя и более категориями приведет к выбору с более высокой вероятностью, по сравнению с бинарными переменными;
- «жадный» (greedy) алгоритм тренировки деревьев решений (он выполняется почти во всех случаях) не гарантирует формирования оптимального дерева решений, потому что в каждом пункте разделения создается локаль-

но оптимальное, а не глобально оптимальное решение. В действительности процедура тренировки глобально оптимального дерева решений представляет собой NP-полную задачу[1].

Леса деревьев решений

Ансамбль (ensemble) означает объединение нескольких классификаторов, при котором создается более сложный и в большинстве случаев более эффективный классификатор. Объединение деревьев решений в ансамбли – многократно проверенная методика создания классификаторов высокого качества. Такие ансамбли также часто называют лесами деревьев решений (decision forests). На практике наиболее часто используется два типа лесов: случайные леса (random forests) и деревья решения с ускорением, или бустингом градиента (gradient-boosted decision trees):

- случайные леса (random forests) формируются как простые ансамбли из нескольких деревьев решений, обычно содержащие от десятков до тысяч таких деревьев. После тренировки каждого отдельного дерева решений обобщенные прогнозы случайных лесов выполняются с учетом статистической моды (типичности) отдельных прогнозов от классификационных деревьев (т. е. каждое дерево «голосует» за свой вариант прогноза) и статистического среднего значения (меаны) прогнозов отдельных деревьев для регрессионных деревьев.

  Следует отметить, что простое наличие множества деревьев решений в лесу приведет к тому, что деревья будут очень похожи друг на друга и будет возникать огромное количество повторяющихся разделений в различных деревьях, особенно для тех признаков, которые являются самыми строгими критериями прогноза для зависимой переменной. Алгоритм формирования случайного леса решает эту проблему, используя следующий алгоритм тренировки.

  1. Для тренировки каждого отдельного дерева случайным образом выбирается подмножество из N элементов (выборка) из общего тренировочного набора данных.
  2. В каждой точке разделения случайным образом выбирается m признаков из p доступных признаков[2], при этом $m \leqslant p$. Далее выбирается оптимальная точка разделения из этих m признаков[3].
  3. Шаг 2 повторяется до тех пор, пока не завершится тренировка отдельного дерева.

---

[1] *Laurent Hyafil and R. L. Rivest.* Constructing Optimal Binary Decision Trees is NP-Complete. Information Processing Letters 5:1 (1976): 15–17.

[2] Для задач классификации с общим количеством признаков p рекомендуется значение $m = \sqrt{p}$. Для задач регрессии рекомендуется значение $m = p/3$. Более подробно об этом *см*. раздел 15.2 книги The Elements of Statistical Learning, 2nd ed., by Trevor Hastie, Robert Tibshirani and Jerome Friedman.

[3] Кроме того, существуют варианты случайных лесов с ограничением набора доступных признаков для каждого отдельного дерева решений. Например, если общий набор признаков {A,B,C,D,E,F,G}, то все точки разделения, создаваемые в дереве решений 1, могут случайным образом выбирать только три признака из ограниченного подмножества признаков {A,B,D,F,G}.

4. Шаги 1, 2 и 3 повторяются до тех пор, пока полностью не завершится тренировка всех деревьев в лесу.

В одиночных деревьях решений часто возникает проблема переподгонки при работе с тренировочными наборами данных, а случайные леса смягчают этот нежелательный эффект, усредняя характеристики множества деревьев решений, что обычно позволяет улучшить эффективность модели. Кроме того, поскольку каждое дерево в случайном лесу может тренироваться независимо от всех прочих деревьев, становится очевидной возможность распараллеливания тренировочного алгоритма. Таким образом, случайные леса способны выполнять процесс тренировки весьма эффективно. Но возрастающая с увеличением размера сложность случайных лесов может существенно увеличить объем ресурсов, требуемых для хранения данных, и значительно затруднить объяснение прогнозов, по сравнению с одиночными деревьями решений;

- деревья решений с ускорением, или бустингом градиента (gradient-boosted decision trees – GBDT), применяют более изощренные комбинации прогнозов отдельных деревьев решений для формирования улучшенных обобщенных прогнозов. При использовании методики ускорения или бустинга градиента несколько слабых обучаемых объектов выборочно объединяются с помощью выполнения оптимизации градиентного спуска в функции потерь, чтобы получить в результате намного более мощную модель обучения. Основной методикой ускорения, или бустинга градиента (gradient boosting) является добавление отдельных деревьев в лес по одному с использованием процедуры градиентного спуска (gradient descent) для минимизации потерь при добавлении деревьев. Процедура добавления деревьев в лес останавливается при достижении установленного предельного количества, когда валидационный набор потерь достигает приемлемого уровня или если дальнейшее добавление деревьев уже не может улучшить (минимизировать) уровень потерь.

  Основная методика GBDT была несколько усовершенствована с целью улучшения производительности, повышения уровня обобщения и создания более эффективных моделей. Рассмотрим подробнее некоторые из этих усовершенствований.

  1. Методика ускорения градиента требует слабых обучаемых объектов. Установление искусственных ограничений для деревьев, таких как предельная глубина дерева, максимальное количество узлов в одном дереве или минимальное количество элементов выборки для одного узла, может помочь в ограничении возможностей таких деревьев без чрезмерного снижения их способностей к обучению.
  2. Может случиться так, что деревья решений, добавляемые на ранней стадии аддитивной тренировки ансамблей с ускорением градиента, вносят гораздо больший вклад в общий прогноз, чем деревья, добавляемые на более поздней стадии этого процесса. В результате формируется несбалансированная модель, сводящая к минимуму все преимущества ансамблирования. Для устранения этой проблемы применяется взвешенная оценка вклада каждого дерева (weighted contribution of each tree), чтобы замедлить процесс обучения. При этом используется методика под на-

званием «сокращение» (shrinkage) для снижения воздействия отдельных деревьев, чтобы позволить деревьям, добавляемым на более поздней стадии, улучшать характеристики модели.

3. Можно объединять стохастические свойства случайных лесов с методикой ускорения градиента при помощи прореживания набора данных (создания дополнительной подвыборки) перед созданием дерева и прореживания набора признаков перед созданием разветвления.
4. Можно воспользоваться стандартными и широко применяемыми методиками регуляризации, такими как L_1- и L_2-регуляризация, для выравнивания конечных весовых коэффициентов обучения, чтобы в дальнейшем избежать переподгонки.

XGBoost[1] – широко известное средство практического применения GBDT, позволяющее достигать превосходных результатов с сохранением возможности правильного масштабирования для больших наборов данных. Как алгоритм, служащий основанием для многих прогрессивных идей в конкурентной борьбе в сфере машинного обучения, эта методика заслужила особое внимание сообщества и стала надежным решением, выбираемым многими специалистами-практиками для создания леса деревьев решений (**https://github.com/dmlc/xgboost**). Но все же GBDT в большей степени подвержены проблеме переподгонки, чем обычные случайные леса, а также более сложны для распараллеливания, поскольку используют аддитивную тренировку, которая полагается на результаты конкретно взятого дерева при обновлении градиента для следующего дерева. Проблему переподгонки в GBDT можно несколько смягчить с помощью упомянутой выше методики сокращения. Кроме того, можно организовать параллельное обучение в отдельном дереве вместо распараллеливания нескольких деревьев.

Метод опорных векторов

Как и логистическая регрессия, метод опорных векторов (support vector machine – SVM) в своей простейшей форме представляет собой линейный классификатор, т. е. позволяет создать гиперплоскость в векторном пространстве, чтобы попытаться разделить два класса в заданном наборе данных. Различием между логистической регрессией и методом опорных векторов является функция потерь. Логистическая регрессия использует функцию логарифмического правдоподобия, которая пропорционально штрафует все элементы (точки) за ошибку в оценке вероятности, даже если эти точки находятся на правильной стороне относительно гиперплоскости. Метод опорных векторов использует функцию зависимых потерь (hinge loss), которая штрафует только те точки, которые расположены на неправильной стороне относительно гиперплоскости или очень близки к гиперплоскости, но находятся на правильной стороне.

Более точно, SVM-классификатор пытается найти максимальную гиперплоскость (maximum-margin hyperplane), разделяющую два класса, где «граница» (margin) обозначает расстояние от разделяющей плоскости до ближайших точек, дан-

[1] *Tianqi Chen and Carlos Guestrin.* XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2016): 785–794.

ных на каждой стороне. В том случае, когда данные разделены не прямой линией, точки внутри этой «границы» штрафуются пропорционально их удаленности от границы. На рис. 2.7 показан конкретный пример: два класса представлены белыми и черными точками. Сплошная линия – это разделяющая плоскость, штриховые линии – границы. Квадратными точками обозначены опорные векторы (support vectors), т. е. векторы (в простейшем случае точки), вносящие ненулевой вклад в функцию потерь. Функция потерь выражается следующей математической формулой:

$$\beta + C\sum_{i=1}^{N}\xi_i,$$

где β – граница, ξ_i – расстояние от i-го опорного вектора до границы, C – гиперпараметр модели, который определяет относительный вклад двух членов этой формулы.

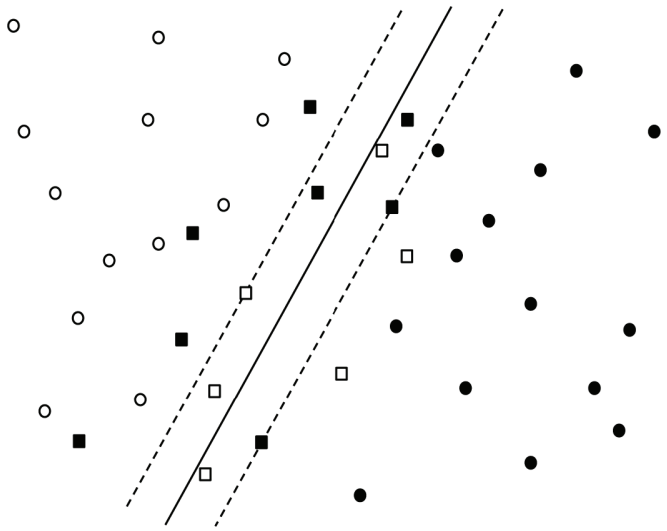


Рис. 2.7 ❖ Граница классификации (сплошная линия) и границы (штриховые линии) для линейного SVM-разделения двух классов (черные и белые точки); квадратами обозначены опорные векторы

Чтобы классифицировать новую точку данных x, нужно просто определить, на какую сторону относительно разделяющей плоскости она помещается. Если необходимо получить оценку реального значения, то можно вычислить расстояние от точки x до разделяющей плоскости, затем применить сигмоиду (сигмоидную функцию) для отображения в интервал [0, 1].

Действительная мощь метода опорных векторов заключается в трюке с ядром (kernel trick), представляющем собой математическое преобразование, которое на входе принимает линейную границу решения и на выходе выдает нелинейную границу. На самом высоком уровне ядро выполняет преобразование одного векторного пространства V_1 в другое векторное пространство V_2. С математической точки зрения ядро является функцией от $V_1 \times V_1$, определяемой как K(x, y), при этом

каждое значение $x \in V_1$ отображается в функцию $K(x,\cdot)$, а V_2 – пространство, охватываемое всеми этими функциями. Например, можно восстановить линейное SVM-пространство, определив функцию K как обычное произведение переменных $K(x, y) = x \cdot y$.

Если ядро нелинейное, то линейный классификатор в пространстве V_1 будет преобразован в нелинейный классификатор в пространстве V_2. Наиболее частым выбором является радиально-базисная функция (radial basis function – RBF) $K(x, y) = e^{-\gamma|x-y|}$. Несмотря на то что здесь мы опускаем математические подробности[1], можно представить себе SVM с ядерной функцией RBF как результат своего рода сглаженной линейной комбинации сфер вокруг каждой точки x. При этом пространство внутри каждой такой сферы классифицируется также как x, а пространство вне сферы относится к противоположному классу. Параметр γ определяет радиус сферы, т. е. максимальное расстояние, на котором должна находиться точка, чтобы классифицироваться как точка центра сферы. Параметр C определяет степень «сглаживания». Большое значение параметра C приведет к тому, что классификатор будет состоять из объединения сфер, тогда как малое значение C дает более извилистую границу, на которую в той или иной степени влияет каждая сфера. Таким образом, можно понять, что при слишком большом значении параметра C создается модель с переподгонкой тренировочных данных, а при слишком малом значении C классификатор будет слабым с точки зрения точности. (Отметим, что параметр γ для каждого RBF-ядра является единственным в своем роде, неповторящимся, в то время как параметр C определяет одни и те же свойства для любого ядра, в том числе и для линейного метода опорных векторов. Оптимальные значения этих параметров обычно вычисляются с использованием поиска по решетке (grid search).)

На практике метод опорных векторов продемонстрировал высокую эффективность, особенно в пространствах с большой размерностью. Тот факт, что такие пространства могут быть описаны в терминах опорных векторов, подтверждает возможность эффективных реализаций этого метода для оценки новых точек данных. Тем не менее сложность тренировки при использовании метода опорных векторов с ядром возрастает в квадратичной зависимости от числа элементов в тренировочных выборках. Таким образом, если размер тренировочного набора превышает несколько миллионов, то ядра используются крайне редко, а граница решения линейна. Другой недостаток состоит в том, что числовые оценки, получаемые методом опорных векторов, нельзя интерпретировать как вероятности. Преобразование итоговых оценок в вероятности требует дополнительных вычислений и перекрестной проверки (кросс-валидации), например применения масштабирования (или калибровки) Платта (Platt scaling) или изотонной регрессии (isotonic regression). Более подробную информацию об этом можно найти в документации библиотеки scikit-learn (**https://scikit-learn.org/stable/modules/calibration.html**).

Наивный байесовский классификатор

Наивный байесовский классификатор (Naive Bayes classifier) является одним из старейших статистических классификаторов. Этот классификатор называется

[1] См. раздел 5.8 и главу 12 книги The Elements of Statistical Learning by Trevor Hastie, Robert Tibshirani and Jerome Friedman.

«наивным», потому что основан на весьма строгих статистических исходных предпосылках, а именно: признаки выбираются независимо из некоторого (неизвестного заранее) распределения. В действительности такое исходное предположение никогда не встречается в реальной жизни. Например, рассмотрим классификатор спама, для которого признаками являются слова в сообщении. Исходное предположение наивного байесовского классификатора утверждает, что сообщение со спамом состоит из независимо выбираемых слов, при этом каждое слово w может быть выбрано с вероятностью $p_{w,spam}$. Аналогичная исходная предпосылка делается для обычных, «правильных» сообщений. Такое исходное предположение совершенно абсурдно, поскольку прежде всего оно полностью игнорирует порядок слов в тексте. Но, даже несмотря на эту очевидную нелепость исходного предположения, наивные байесовские классификаторы продемонстрировали достаточно высокую эффективность при решении таких задач, как классификация спама.

Основная идея, заложенная в основу наивного байесовского классификатора, заключается в следующем: взяв элемент (точку) данных из набора признаков $X = x_1, \ldots, x_n$, необходимо определить вероятность того, что метка Y для этого элемента представляет класс C. В формуле 2.1 эта концепция выражена как условная вероятность:

$$\Pr[Y=C \mid X = (x_1, \ldots, x_n)]. \tag{2.1}$$

Теперь, используя теорему Байеса, эту вероятность можно представить в другом виде, показанном в формуле 2.2:

$$\frac{\Pr[X = (x_1,\ldots,x_n) \mid Y = C] \cdot \Pr[Y = C]}{\Pr[X = (x_1,\ldots,x_n)]}. \tag{2.2}$$

Если принять весьма строгое исходное предположение о том, что для выборки данных в каждом классе признаков элементы выбираются независимо друг от друга, то получим формулу 2.3:

$$\Pr[X = (x_1,\ldots,x_n) \mid Y = C] = \prod_{i=1}^{n} \Pr[X_i = x_i \mid Y = C]. \tag{2.3}$$

Можно вычислить оценку числителя в формуле 2.2 по данным с метками: $\Pr[X_i = x_i \mid Y = C]$ – это просто часть элементов выборки с i-м признаком, равным x_i, из всех элементов выборки класса C, при этом $\Pr[Y = C]$ – это часть элементов выборки класса C из всех помеченных элементов выборки.

А что можно сказать о знаменателе формулы 2.2? Оказывается, нет необходимости его вычислять, потому что при классификации по двум классам вполне достаточно вычислить отношение (ratio) оценок вероятности для двух классов C_1 и C_2. Это отношение (формула 2.4) дает оценку в форме положительного действительного числа:

$$\begin{aligned} \theta &= \frac{\Pr[Y = C_1 \mid X = (x_1,\ldots,x_n)]}{\Pr[Y = C_2 \mid X = (x_1,\ldots,x_n)]} \\ &\approx \frac{\Pr[Y = C_1] \prod_{i=1}^{n} [X_i = x_i \mid Y = C_1]}{\Pr[Y = C_2] \prod_{i=1}^{n} [X_i = x_i \mid Y = C_2]}. \end{aligned} \tag{2.4}$$

Оценка $\theta > 1$ означает, что C_1 является более вероятным классом, тогда как оценка $\theta < 1$ говорит о том, что более вероятен класс C_2. (Если необходима оптимизация либо по точности, либо по критерию изъятия, можно выбрать другое пороговое значение для границы классификации.)

Внимательный читатель заметит, что оценка θ вычислена без обращения к функции потерь или к алгоритму оптимизации. В действительности алгоритм оптимизации скрыт в выражении оценки вероятности $Pr[X_i = x_i \mid Y = C]$. Использование части элементов выборки из набора тренировочных данных дает оценку по методу максимального правдоподобия (maximum likelihood estimate), т. е. ту же функцию потерь, которая применяется для логистической регрессии. На этом сходство с логистической регрессией не заканчивается: если взять логарифмы выражений в формуле 2.4, то правая часть становится линейной функцией исследуемых признаков. На этом основании наивный байесовский метод можно рассматривать еще и как линейный классификатор.

При использовании наивной байесовской классификации на практике необходимо учитывать следующие тонкости:

- что происходит, когда все экземпляры некоторого признака принадлежат к одному и тому же классу (например, торговые марки медицинских средств для улучшения потенции появляются только в спам-сообщениях)? При этом одна из частей формулы 2.4 будет равна нулю, в результате оценка θ становится равной нулю или бесконечности, что совершенно не имеет смысла. Для устранения этой проблемы используется сглаживание (smoothing), при котором добавляются фиктивные («фантомные») элементы в помеченные данные для каждого признака. Например, если сглаживание выполняется с помощью множителя α, то значение для любого признака должно вычисляться следующим образом:

$$Pr[X_i = x_i \mid Y = C] = \frac{(\text{число элементов выборки в классе C с признаком } x_i) + \alpha}{(\text{число элементов выборки в классе C}) + \alpha \cdot (\text{число признаков})}.$$

  При выборе множителя $\alpha = 1$ метод называется сглаживанием Лапласа (Laplace smoothing), а если множитель $\alpha < 1$, то это сглаживание Лидстоуна (Lidstone smoothing);
- что происходит, если признак x_i появляется в валидационном (проверочном) наборе (или еще хуже того – в реальном оцениваемом наборе данных), но ранее он не появлялся в тренировочном наборе данных? В этом случае мы вообще не получим оценку для $Pr[X_i = x_i \mid Y = C]$. Простейшая оценка должна быть определена как вероятность для $Pr[Y = C]$. Для практического применения более сложных подходов необходимо обратиться к работе Фримана[1].

Последнее замечание: можно отобразить оценку $\theta \in (0, \infty)$ в вероятность в интервале (0, 1), используя формулу отображения $\theta \to \theta/(1 + \theta)$, но такая оценка вероятности будет неправильно откалибрована. Как и в методе опорных векторов, для получения более качественных оценок вероятности рекомендуется применение таких методик, как калибровка (масштабирование) Платта или изотонная регрессия.

---

[1] *David Freeman.* Using Naive Bayes to Detect Spammy Names in Social Networks. Proceedings of the 2013 ACM Workshop on Artiicial Intelligence in Security (2013): 3–12.

Метод k ближайших соседей

Алгоритм k ближайших соседей (k-nearest neighbors – k-NN) является самым широко известным примером методики ленивого обучения (lazy learning). Этот тип методики машинного обучения откладывает большую часть вычислений на время классификации, вместо того чтобы выполнять эту работу во время тренировки. Модели ленивого обучения не обучаются обобщениям данных во время тренировочной стадии. Вместо этого они фиксируют (записывают) все передаваемые им точки тренировочных данных и используют эту информацию для создания локальных обобщений на тестовой выборке во время классификации. Метод k ближайших соседей является одним из самых простых алгоритмов машинного обучения со следующими характеристиками:

- этап тренировки состоит из простой записи (фиксации) всех векторов признаков и соответствующих элементов меток в этой модели;
- прогноз классификации[1] – это просто наиболее часто встречающаяся метка среди k ближайших соседей в тестовой выборке (в соответствии с наименованием методики).

Метрики расстояния для определения того, насколько «близко» точки расположены друг от друга в n-мерном пространстве признаков (где n – размер векторов признаков), обычно представляют собой евклидову метрику (расстояние) для непрерывных переменных и расстояние Хэмминга для дискретных переменных.

Можно сразу предположить, что при таком простом алгоритме этап тренировки k ближайших соседей обычно вполне сравним с другими алгоритмами обучения в отношении затрат времени обработки при классификации. Кроме того, тот факт, что все векторы признаков и метки необходимо хранить в модели, приводит к созданию весьма неэффективной с точки зрения использования ресурсов памяти модели (space-inefficient model). (Модель k-NN, которая требует 1 Гб тренировочных векторов признаков, будет иметь размер не менее 1 Гб.)

Благодаря простоте метод k-NN часто используют в качестве практического примера для ознакомления начинающих с концепциями машинного обучения, но на практике эта методика встречается редко из-за серьезных недостатков, перечисленных ниже:

- слишком большие размеры модели, так как в моделях обязательно должны храниться (как минимум) все тренировочные данные векторов признаков и меток;
- медленная скорость классификации, потому что вся работа по обобщению выполняется до начала времени классификации. Поиск ближайших соседей может потребовать значительных затрат времени, особенно если модель хранит элементы тренировочных данных способом, не оптимизированным для пространственного поиска. Деревья k-d (подробно рассматриваемые в разделе «Деревья k-d» ниже) часто используются как оптимизированная структура данных для ускорения поиска соседей[2];

---

[1] Алгоритм k-NN также можно использовать для регрессии – обычно k ближайших соседних меток элементов тестовой выборки данных берутся для прогнозирования результата.

[2] *Jon Louis Bentley.* Multidimensional Binary Search Trees Used for Associative Searching. Communications of the ACM 18 (1975): 509–517.

- высокая чувствительность к несбалансированности классов[1] в исследуемом наборе данных. Классификации отклоняются в сторону классов с бóльшим количеством элементов выборки в тренировочных данных, так как наиболее вероятно, что элементы этих классов будут включены в набор k-NN при любой произвольно взятой тестовой выборке данных;
- низкая точность классификации из-за высокого уровня шума в данных, неполных или немасштабированных признаков (выбор более высокого значения k снижает влияние шума в тренировочных данных, но при этом может ослабить процесс обучения);
- трудность выбора правильного значения параметра k. Результаты классификации в очень высокой степени зависят от этого параметра, и выбор такого значения k, которое успешно работает во всех частях пространства признаков, может вызывать затруднения из-за различных плотностей распределения в исследуемом наборе данных;
- потеря работоспособности при высоких размерностях из-за «проклятия размерности» (curse of dimensionality). Кроме того, при высоких размерностях пространства признаков область «соседства» для любой произвольной точки становится слишком большой, что приводит к возникновению шумов (помех) при выборе соседей.

Нейронные сети

Искусственные нейронные сети (artificial neural networks – ANN) представляют целый класс методов машинного обучения, к которым в последнее время в очередной раз проявляется повышенное внимание. Историю создания нейронных сетей можно проследить до 1942 года, когда Маккаллок (McCulloch) и Питтс (Pitts) опубликовали свою действительно революционную работу, в которой сформулировали основные положения о том, как работает нервная система человека[2]. После этого до 1970-х гг. исследования в области нейронных сетей выполнялись весьма медленными темпами в основном из-за того, что широкую известность приобрела концепция вычислительных архитектур фон Неймана (von Neumann), абсолютно противоположная концепции ANN. Даже после возрождения интереса к этой области в 1980-х гг. интенсивность исследований оставалась низкой, поскольку высокие требования к вычислительным ресурсам для тренировки нейронных сетей вынуждали исследователей ждать результатов своих экспериментов в течение нескольких дней или даже недель. Недавний всплеск интереса к нейронным сетям был вызван сочетанием технологических достижений в области аппаратуры, а именно использованием графических процессоров (GPU) для «в определенной степени чудесного» распараллеливания и ускорения процесса тренировки искусственных нейронных сетей, а также появлением возможности использования огромных объемов данных, которые требуются искусственным нейронным сетям для достижения приемлемых результатов при решении сложных задач, таких как распознавание образов и распознавание речи.

---

[1] Концепция несбалансированности классов более подробно будет рассматриваться в главе 5.

[2] *W. S. McCulloch and W. H. Pitts.* A Logical Calculus of Ideas Immanent in Nervous Activity. Bulletin of Mathematical Biophysics 5 (1942): 115–133.

В человеческом мозге содержится колоссальное число нейронов (порядка 10 млрд), каждый из которых соединен с десятками тысяч других нейронов. Каждый нейрон принимает электрохимические входные сигналы от других нейронов, и если сумма этих электрических входных сигналов превышает определенный уровень, то нейрон включает на выходе режим передачи другого электрохимического сигнала нейронам, которые соединены с ним. Если входные сигналы не превышают этот уровень, то нейрон не генерирует выходной сигнал. Каждый нейрон представляет собой очень простое обрабатывающее устройство, обладающее весьма ограниченной функциональностью, но сочетание с огромным количеством других нейронов, соединенных в разнообразных формах и слоях, обеспечивает способность мозга выполнять чрезвычайно сложные задачи: от различения кошек и собак до восприятия глубоких философских концепций.

В начальной стадии исследований с помощью искусственных нейронных сетей пытались смоделировать нейроны в человеческом мозге, чтобы выйти на уровень обучения, подобный человеческому. Отдельные нейроны моделировались с помощью простых математических ступенчатых функций (называемых функциями активации (activation functions)), принимающих на входе взвешенные сигналы (данные) от нескольких нейронов и выдающих выходные сигналы для нескольких других нейронов, если достигался уровень активации. Такую математическую модель биологического нейрона также называли перцептроном (perceptron). С помощью перцептронов можно формировать бесконечное множество различных нейронных сетей, изменяя топологию, функции активации, цели обучения или методы тренировки модели.

Обычно искусственные нейронные сети состоят из нейронов, распределенных по слоям (layers). Каждый нейрон в слое принимает входные данные из предыдущего слоя и, если произошла его активация, генерирует выходные данные для одного или нескольких нейронов в следующем слое. Каждому соединению между двумя нейронами присвоена весовая характеристика (weight), кроме того, каждый нейрон или слой также может обладать собственной характеристикой отклонения (bias). Эти параметры должны пройти тренировочное обучение с помощью процесса обратного распространения ошибки (backpropagation), который будет описан ниже в краткой и упрощенной форме. Перед началом тренировки все весовые параметры и отклонения инициализируются случайным образом. Для каждого элемента выборки в тренировочном наборе выполняется два шага[1]:

1) проход вперед. Продвижение входных данных через искусственную нейронную сеть и получение текущего прогноза;
2) проход в обратном направлении. Если прогноз правильный, то поощряются соединения, сгенерировавшие этот результат, т. е. увеличиваются их весовые характеристики пропорционально вкладу в правильный прогноз. Если прогноз неверный, то штрафуются соединения, которые привели к этому неправильному результату.

Нейронные сети в течение многих лет широко использовались в промышленности и поддерживались значительным объемом научных исследований. Сущест-

[1] Это чрезвычайно упрощенное описание работы искусственных нейронных сетей, которое позволяет получить лишь самое общее представление. Более полную информацию по этой теме можно найти в книге Deep Learning by Ian Goodfellow, Yoshua Bengio and Aaron Courville (MIT Press).

вуют сотни вариантов инфраструктуры нейронных сетей, которые можно использовать как при обучении с учителем, так и при обучении без учителя. Важным свойством некоторых искусственных нейронных сетей является то, что они могут выполнять обучение признакам без учителя (unsupervised feature learning) (отличается от обычного обучения без учителя), при котором требуется минимальное применение конструирования признаков или не требуется вовсе. Например, создание классификатора вредоносных программ с помощью метода опорных векторов требует привлечения экспертов в проблемной области для конструирования признаков (это будет подробно рассматриваться в главе 4). При использовании искусственных нейронных сетей можно просто передавать в сеть собственно инструкции процессора или граф вызовов, а сеть сама определит, какие признаки являются существенными для задачи классификации.

Даже несмотря на существование огромного разнообразия аппаратных и программных оптимизаций для тренировки нейронных сетей, все же они остаются значительно более затратными с точки зрения использования вычислительных ресурсов, чем, например, тренировка дерева решений. (С другой стороны, прогнозы могут выполняться более эффективно благодаря многослойной структуре.) Количество гиперпараметров модели для тонкой настройки процесса тренировки искусственных нейронных сетей может быть огромным. Например, необходимо обязательно выбрать конфигурацию архитектуры сети, тип функции активации, требуется ли полное соединение нейронов или оставить слои слабо соединенными и т. д. Наконец, несмотря на то что искусственные нейронные сети, или сети глубокого обучения (deep learning networks), как неформально называют глубокие (многослойные) варианты нейронных сетей, часто считают надежным универсальным решением в области машинного обучения, они могут быть весьма запутанными и сложными для объяснения и обоснования. Нередко выбираются альтернативные решения (например, другие алгоритмы, описанные в этом разделе), которые работают быстрее и более понятны и просты для тренировки, освоения и объяснения.

Практические аспекты классификации

С теоретической точки зрения применение алгоритма обучения вполне понятно: тренировочные данные помещаются в большую матрицу (которую называют матрицей плана (design matrix)), запускается процедура тренировки, затем полученная модель используется для классификации неизвестных данных. Но после начала кодирования обнаруживается, что задача не так проста. Существуют десятки вариантов выбора, доступных в процессе формирования модели, и каждый выбор может привести к получению совершенно различных итоговых результатов работы окончательно сформированной модели. Рассмотрим некоторые из наиболее важных вариантов выбора, которые доступны в процессе создания модели.

Выбор семейства моделей

В предыдущем разделе рассматривался ряд различных алгоритмов классификации с учителем. Но как принять решение о выборе одного из этих алгоритмов для решения конкретно поставленной задачи? Наилучший ответ на этот вопрос: позволить самим данным сделать выбор, т. е. попробовать несколько различных

методик, чтобы понять, какая из них работает лучше остальных. Если для таких проб недостаточно времени или отсутствует необходимая инфраструктура, то рекомендуется принять во внимание следующие факторы:

- сложность вычислений: для тренировки выбранной модели потребуется приемлемое количество времени, при этом используются все имеющиеся данные. Модели логистической регрессии и линейный метод опорных векторов можно натренировать очень быстро. Кроме того, для логистической регрессии и лесов деревьев решений существуют весьма эффективные реализации с распараллеливанием, способные обработать очень большие объемы данных. Для метода опорных векторов с ядром и нейронных сетей может потребоваться достаточно длительное время тренировки;
- математическая сложность: возможно, для имеющегося набора данных вполне подойдет линейная граница решения, обеспечивающая хороший уровень классификации, но большинство наборов данных имеет нелинейные границы. Логистическая регрессия, линейный метод опорных векторов и наивная байесовская классификация – линейные алгоритмы. Для нелинейных границ можно воспользоваться лесами деревьев решений, методом опорных векторов с ядром или нейронными сетями;
- объяснимость: может потребоваться изучение результатов работы выбранной модели человеком, чтобы определить, почему было принято именно такое решение. Например, если блокируется законопослушный пользователь, необходимо объяснить ему, какие подозрительные действия он совершил. Наилучшим семейством моделей с точки зрения объяснимости является дерево решений, так как оно точно определяет причины выполненной классификации. Можно воспользоваться относительными весовыми характеристиками признаков для объяснения результатов логистической регресии и наивной байесовской классификации. Результаты работы лесов деревьев решений, метода опорных векторов и нейронных сетей объяснить очень трудно.

Существует множество различных мнений о том, какую модель следует использовать в той или иной конкретной ситуации. Поиск в интернете по фразе «machine learning cheat sheet» дает десятки результатов. Можно выделить основную рекомендацию: использовать леса деревьев решений для высокой точности результатов при регулируемом (управляемом) количестве признаков (до нескольких тысяч), логистическую регрессию для быстрой тренировки при очень большом количестве признаков (десятки тысяч и более). Но в любом случае лучше всего поэкспериментировать, чтобы найти наилучший вариант для имеющихся конкретных данных.

Формирование процесса тренировки данных

При решении задачи обучения с учителем требуется каким-либо образом собрать помеченные экземпляры тех данных, которые необходимо классифицировать, – регистрационные данные учетных записей, системные имена пользователей, сообщения электронной почты и т. п. Поскольку целью является выбор модели, которая может прогнозировать будущие результаты на основе уже имеющихся предыдущих данных, необходимо выделить часть помеченных данных для работы с ними как с «будущими» данными, чтобы можно было оценить качество работы модели. Это можно сделать различными способами, описанными ниже:

- кросс-валидация (cross-validation) – это стандартная методика для оценки моделей в тех случаях, когда тренировочных данных не очень много, поэтому каждый помеченный экземпляр вносит значительный вклад в модель обучения. При использовании этой методики помеченные данные разделяются на k равных частей (обычно от 5 до 10) и тренируются k различных моделей: каждая модель «игнорирует» одну из k частей данных (игнорируемые части не должны быть одинаковыми у разных моделей) и тренируется на остальных k–1 частях. В дальнейшем игнорируемая часть используется для валидации (проверки). В итоге k различных моделей объединяются по усредненным статистическим данным о производительности и параметрам модели;
- тренировка/валидация/тестирование (train/validate/test): эта методика наиболее часто применяется при наличии достаточного объема тренировочных данных, так что исключение половины этих данных не оказывает существенного воздействия на качество работы модели. При использовании данной методики все помеченные данные случайным образом делятся на три части (*см*. схему слева на рис. 2.8). Бóльшая часть (например, 60 %) становится тренировочным набором (training set), результатом работы с которым является алгоритм обучения. Вторая часть (например, 20 %) – валидационный набор (validation set), который используется для оценки и итеративной проверки результатов работы модели по алгоритму обучения. Например, можно более точно настроить параметры модели на основе оценки производительности при работе с валидационным набором. Наконец, после выбора оптимальной модели используется оставшаяся часть данных (в нашем примере 20 %) как тестовый набор для оценки реальной производительности на новых данных;
- разновременная валидация (out-of-time validation): эта методика позволяет решить задачу валидации (проверки) моделей, когда распределение данных изменяется со временем. Здесь случайные выборки тренировочного и валидационного наборов из одного и того же массива помеченных данных являются в определенном смысле «нечестными» – распределения тренировочного и валидационного наборов могут совпадать в высокой степени и не отображают фактор времени. Более правильным подходом является разделение тренировочного и валидационного наборов на основе отсечек времени: выборки до определенного времени t представляют тренировочный набор, а выборки после определенного времени t – валидационный набор (*см*. схему справа на рис. 2.8). Может потребоваться тестовый набор, сформированный в тот же интервал времени, что и валидационный набор, или в следующий за ним интервал времени.

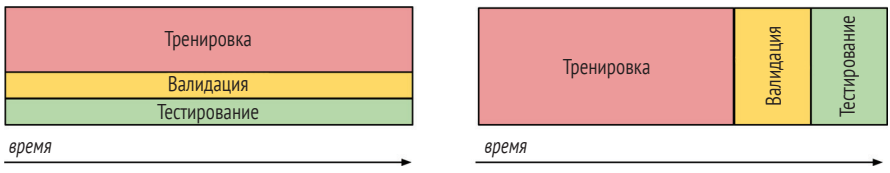


Рис. 2.8 ❖ Одновременная валидация (схема слева) и разновременная валидация (схема справа)

Формирование тренировочного набора данных также может потребовать особого внимания. Рассмотрим несколько общих проблем.

Несбалансированные данные

При попытке классификации редкого события/явления, например нелегального захвата учетной записи, «корректные» элементы выборки могут составлять 99 % или даже 99,9 % от всех помеченных данных. В этом случае «некорректные» элементы могут не иметь достаточного веса для воздействия на классификатор, поэтому модель, возможно, продемонстрирует весьма низкую эффективность при тренировке на таких чрезвычайно не сбалансированных данных. Для устранения этой проблемы существует несколько возможностей:

- выборка с запасом (oversampling) – повышение доли класса меньшинства в выборке. Другими словами, повторение фактов обнаруженных «некорректностей» в тренировочном наборе данных для создания более сбалансированного набора;
- субдискретизация (undersampling) – понижение доли класса большинства в выборке, т. е. выбор случайного подмножества класса большинства для создания более сбалансированного набора;
- изменение функции потерь для улучшения (увеличения) весовой характеристики класса меньшинства, для того чтобы каждый элемент меньшинства оказывал большее воздействие на модель.

Существуют различные мнения по поводу правильного формирования баланса классов для тренировки при классификации редких событий. Главная проблема заключается в компромиссе между обучением на редких событиях и априорным обучением, т. е. редкое событие действительно происходит редко. Например, если выполняется тренировка на выборке с 50 % редких событий, то в результате обучения модель может решить, что такое событие должно происходить в течение половины времени. Необходимы некоторые эксперименты для определения оптимального баланса.

В любом случае, если выборка тренировочных данных производится искусственным путем, то должна быть уверенность в том, что валидационный и тестовый наборы данных не подбираются искусственно, или необходимо использовать метрику эффективности, которая не изменяется при любом способе выборки (например, метрика ROC AUC, описанная ниже в разделе «Выбор пороговых значений и сравнение моделей»). В противном случае искусственная выборка для валидационного набора будет искажать метрики эффективности.

Неучтенные признаки

В идеальном мире каждое событие тщательно документируется с фиксацией точных данных, необходимых для его классификации. В реальной жизни все не так: в журнальных записях появляются ошибки и неточности. Обнаруживается лишь частичный набор данных, необходимых для определения конкретного признака. Данные о некоторых признаках фиксируются с задержкой или вообще «вычищаются» (уничтожаются). В результате в некоторых выборках могут оказаться неучтенные (пропущенные) признаки. Как включить элементы с неучтенными признаками в тренировочный набор?

Один из подходов – простое удаление любого события с неучтенными признаками. Если признаки пропущены из-за случайных единичных сбоев, то такой

подход вполне допустим. Но если данные с неучтенными признаками сконцентрированы около конкретного события или типа данных, то отбрасывание таких данных изменит распределение.

Для использования выборки с неучтенным признаком необходима импутация (imputation) значения неучтенного признака. Существует большое количество литературы по импутации, которую мы здесь даже не пытаемся рассматривать. Достаточно сказать, что это самая простая методика присваивания неучтенному признаку среднего, или медианного, значения этого признака. Более сложные методики включают использование существующих признаков для прогнозирования значения конкретного неучтенного признака.

Крупномасштабные события

В атмосфере постоянной конкуренции возможны крупномасштабные атаки, предпринимаемые относительно не искушенными деятелями, которые можно остановить без особых затруднений. Если вы необдуманно включаете эти события в набор тренировочных данных, то модель может обучиться тому, как останавливать подобные атаки, но оставит без должного внимания менее значительные по объему, но более изощренные и хитроумные атаки. Следовательно, для обеспечения более высокой эффективности, возможно, потребуется снижение в выборке доли крупномасштабных событий.

Развитие способностей атакующих злоумышленников

В среде с высокой конкуренцией атакующие редко отказываются от попыток взлома после развертывания новой защиты, напротив, они усовершенствуют свои методы в попытках обойти средства защиты. Вы должны предпринять ответные меры, атакующие реагируют на них, и это противостояние продолжается бесконечно. Таким образом, распределение атак не только изменяется со временем, оно динамически изменяется в ответ на действия защищающейся стороны. Для создания модели, надежно противостоящей текущим атакам, в тренировочном наборе самым свежим данным должны присваиваться более высокие весовые характеристики с учетом только самых последних n дней или недель или с использованием некоторого типа функции ослабления, чтобы уменьшить значимость более старых данных.

С другой стороны, возникает опасность того, что модель «забудет» атаки, предпринятые в прошлом. В качестве конкретного примера предположим, что каждый день вы тренируете новую модель на актуальных данных, собранных за последние семь дней. В понедельник происходит атака, которую вы не можете остановить (несмотря на то что быстро обнаружили и пометили ее). Во вторник модель выбирает новые помеченные данные и способна остановить такую атаку. В среду атакующий понимает, что он заблокирован и отступает. Теперь посмотрим, что происходит в следующую среду: в последние семь дней не встречались экземпляры этой атаки, поэтому новая модель, возможно, не настроена для ее отражения. Если атакующий обнаружит эту уязвимость, то весь цикл повторится с самого начала.

Все вышеизложенные соображения демонстрируют возможные недостатки и ошибки в конкретных вариантах выбора тренировочных наборов данных. Вы сами должны правильно оценивать компромиссы между актуальностью данных, надежностью хронологически накопленных данных и вычислительной мощностью системы для формирования наилучшего решения в каждом конкретном случае.

Выбор признаков

Если в вашем распоряжении имеется действительно эффективная инфраструктура машинного обучения, то большая часть времени и усилий будет затрачена на конструирование признаков (feature engineering), т. е. на выявление и понимание сигналов, которые можно использовать для идентификации атак. Далее эти сигналы встраиваются в конвейер тренировки и оценки. Чтобы эта работа была наиболее эффективной, необходимо использовать только те признаки, которые обладают характерными свойствами, резко отличающимися от прочих, и позволяют четко определить различия. Добавление каждого признака должно существенно улучшать модель.

Помимо требуемых дополнительных усилий по созданию и сопровождению, избыточные признаки могут ухудшить качество модели. Если число признаков больше количества точек данных, то модель будет переподогнанной: имеется достаточное количество параметров модели для построения кривой по всем тренировочным данным. Кроме того, признаки с сильной корреляцией могут привести к нестабильности решений, принимаемых моделью. Например, если имеется признак «количество регистраций в системе вчера» и признак «количество регистраций в системе за последние два дня», то информация, которую вы пытаетесь собрать, будет разделена между этими двумя признаками совершенно произвольным образом, и модель, вероятнее всего, не сможет понять, какой из этих признаков важнее.

Проблему корреляции признаков можно решить, вычисляя ковариационные матрицы для этих признаков и объединяя признаки с сильной корреляцией (или проецируя их в ортогональные пространства; в предыдущем примере признак «количество регистраций в системе позавчера» стал бы более удачным выбором, чем «количество регистраций в системе за последние два дня»).

Существует несколько методик для решения задачи выбора признаков:

- для логистической регрессии, метода опорных векторов и деревьев/лесов решений имеются методы определения относительной важности признаков. Можно воспользоваться этими методами и сохранить только те признаки, которые наиболее важны;
- можно воспользоваться методом регуляризации L_1 (*см.* следующий раздел) для выбора признаков в классификаторах логистической регрессии и метода опорных векторов;
- если количество признаков n не слишком велико (например, $n < 100$), можно применить подход с постепенным увеличением признаков в модели: создать n моделей с одним признаком и определить, какая из них дает самый лучший результат на валидационном наборе, затем создать $n-1$ моделей с двумя признаками и т. д., до тех пор пока при очередном добавлении дополнительного признака не будет достигнуто определенное пороговое значение;
- можно также использовать противоположный подход с постепенным уменьшением признаков в модели: создать модель с n признаками, затем n моделей с $n-1$ признаками и сохранить самую лучшую. Продолжать до тех пор, пока потери от удаления дополнительного признака не станут слишком велики.

Библиотека scikit-learn предлагает реализацию `sklearn.feature_selection.SelectFromModel` (**https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html**), вспомогательной утилиты, позволяющей выбирать признаки на основе весовых характеристик их важности. Пока в тренируемой функции оценки имеется атрибут `feature_importances_` или `coef_` после подгонки[1], его можно передать в утилиту `SelectFromModel` для ранжирования важности выбираемых признаков. Предположим, что у нас уже есть предварительно натренированная модель `DecisionTreeClassifier` (имя переменной `clf`) и исходный тренировочный набор данных (переменная `train_x`) со 119 признаками. Ниже приведен небольшой фрагмент кода, демонстрирующий использование утилиты `SelectFromModel` для сохранения только признаков, у которых значение `feature_importance` лежит выше меаны[2]:

```
from sklearn.feature_selection import SelectFromModel

sfm = SelectFromModel(clf, prefit=True)

# Генерация нового тренировочного набора, сохранение только выбранных признаков
train_x_new = sfm.transform(train_x)

print("Original num features: {}, selected num features: {}"
      .format(train_x.shape[1], train_x_new.shape[1]))

> Original num features: 119, selected num features: 7
```

Переподгонка и недоподгонка

При использовании любого алгоритма машинного обучения может возникать проблема переподгонки (overfitting). Создаваемая модель настолько хорошо соответствует тренировочным данным, что не способна правильно обобщать новые неизвестные данные. Например, рассмотрим границу решения, показанную для двумерного набора данных в левой части рис. 2.9. Все точки классифицированы правильно, но форма границы слишком сложна, поэтому маловероятно, что такую границу можно будет использовать для эффективного разделения новых точек.

С другой стороны, чрезмерная простота модели также может привести к слабости обобщения новых неизвестных данных. Эта проблема называется недоподгонкой (underfitting). Например, рассмотрим границу решения, показанную в правой части рис. 2.9. Эта простая прямая линия корректна в подавляющем большинстве случаев для тренировочных данных, но становится причиной огромного коли-

---

[1] Большинство функций оценки на основе деревьев, таких как `DecisionTreeClassifier` и `RandomForestClassifier`, а также некоторые ансамбли функций оценки, такие как `GradientBoostingClassifier`, имеют атрибут `feature_importances_`. Обобщенные линейные модели, такие как `LinearRegression` и `LogisticRegression`, и метод опорных векторов, например SVC, имеют атрибут `coef_`, позволяющий утилите `SelectFromModel` сравнивать величины этих коэффициентов или характеристики важности, соответствующие каждому признаку.

[2] Использование меаны как порогового значения важности признака является стратегией, принятой по умолчанию для утилиты `SelectFromModel`, если не задано другое значение параметра `threshold`, например `median`, или статическое значение.
Пример практического применения утилиты sklearn.feature_selection.SelectFromModel для решения реальной задачи можно найти в репозитории кода для этой книги: Python Jupyter notebook **https://github.com/oreilly-mlsec/book-resources/blob/master/chapter2/select-from-model-nslkdd.ipynb**.

чества ошибок и, вероятнее всего, покажет весьма низкую эффективность для новых неизвестных данных.

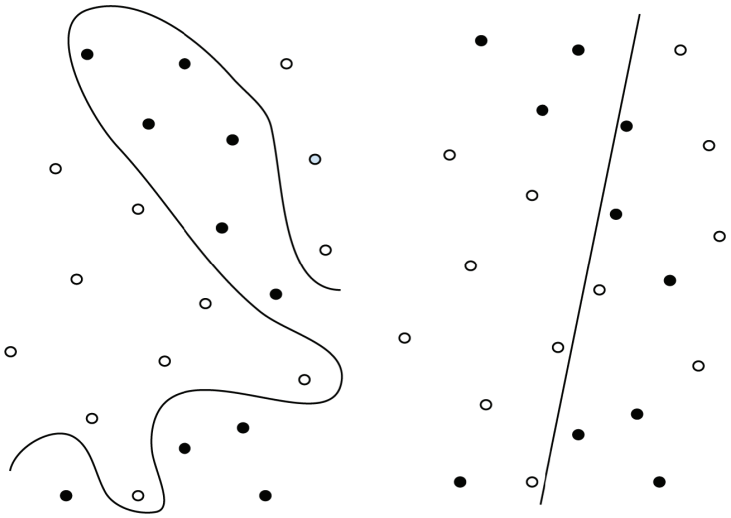


Рис. 2.9 ❖ Слева: переподгонка границы решения. Справа: недоподгонка границы решения

Наиболее часто применяемой методикой для минимизации переподгонки и недоподгонки является включение сложности модели в процедуру тренировки. Такой метод обозначают математическим термином «регуляризация» (regularization), означающим добавление в функцию потерь элемента, который представляет сложность модели в числовом выражении. Если φ представляет модель, y_i – тренировочные метки, а $\hat{y}_i$ – прогнозы (метки или вероятности), то функция потерь с регуляризацией выражается следующей формулой:

$$\mathscr{L}(\varphi) = \sum_i \ell(\hat{y}_i, y_i) + \lambda \cdot \Omega(\varphi),$$

где ℓ – вышеупомянутая обычная функция потерь, а Ω – штрафная функция. Например, в дереве решений Ω может быть числом листьев, т. е. деревья с чрезмерно большим количеством листьев должны быть «оштрафованы». Параметр λ можно настроить для сбалансированного компромисса между обычной функцией потерь и выражением регуляризации. Если значение λ очень мало, то можно получить модель с переподгонкой; если значение λ слишком велико, то результатом может стать модель с недоподгонкой.

В логистической регрессии стандартная регуляризация представляет собой норму вектора коэффициентов $\beta = (\beta_0, \ldots, \beta_n)$. Существуют два способа вычисления нормы: регуляризация L_2 использует стандартную эвклидову норму $|\beta| = \Sigma_i \beta_i^2$, в то время как при регуляризации L_1 используется так называемое «манхэттенское расстояние», или расстояние городских кварталов, $|\beta| = \Sigma_i |\beta_i|$. Норма L_1 имеет следующее свойство: локальные минимумы возникают там, где коэффициенты признаков равны нулю. Таким образом, регуляризация L_1 позволяет выбрать признаки с наибольшим вкладом в модель.

При использовании регуляризированной логистической регрессии необходимо уделить особое внимание нормализации признаков перед тренировкой модели, например применяя для этого линейную трансформацию, в результате которой каждый признак получает меану 0 и стандартное отклонение 1. Если не применяется трансформация подобного рода, то коэффициенты различных признаков не являются сравнимыми. Например, признак «возраст учетной записи в секундах» в самом общем случае будет иметь намного большее значение, чем признак «количество друзей в социальном графе». Следовательно, коэффициент для «возраста» будет гораздо меньшим по сравнению с коэффициентом для количества друзей, для того чтобы обеспечить одинаковый эффект. Регуляризация штрафует коэффициент друзей намного строже.

Вне зависимости от используемой модели необходимо выбрать конкретные параметры регуляризации на основе экспериментальных данных из валидационного набора. Будьте внимательны и осторожны, чтобы не допустить переподгонки валидационного набора. Для этого и существует тестовый набор: если эффективность на тестовом наборе значительно хуже, чем на валидационном наборе, то произошла переподгонка выбранных параметров.

Выбор пороговых значений и сравнение моделей

Выбранный алгоритм классификации (обучения) с учителем обычно выдает оценку в виде действительного числа, а вы должны выбрать пороговое значение (threshold) или несколько пороговых значений[1], превышение которых приводит к блокированию действия или активизации дополнительной функции (например, требование номера телефона). Как выбрать такое пороговое значение? Окончательный выбор – это решение, принимаемое с деловой точки зрения, на основе компромисса между безопасностью и неудобствами для пользователя. В идеальном случае принимается некоторая функция стоимости, например 1 ложноположительный случай равен 10 ложноотрицательным случаям, и выполняется минимизация суммарной стоимости в репрезентативной выборке данных. Другой вариант – коррекция значения точности или полнота цели, например 98 %, и выбор порогового значения, которое позволяет достичь этой цели.

Предположим, что имеются две версии выбранной модели с различными параметрами (например, с различной регуляризацией) или даже из различных семейств моделей (например, логистическая регрессия и случайный лес деревьев решений). Какая версия лучше? Если имеется функция стоимости, то все просто: вычислить стоимость обеих версий на одном наборе данных и выбрать вариант с меньшей стоимостью. Если корректируется точность цели, то выбрать версию, оптимизирующую полноту, и наоборот.

Другим широко используемым методом сравнения моделей является построение кривой рабочей характеристики приемника (РХП) (receiver operating

[1] В большинстве библиотек машинного обучения для этого по умолчанию применяется следующий метод: выбирается класс с наивысшей оценкой, которая используется как результат прогнозирования. Например, для задачи бинарной классификации это просто переводится в пороговое значение 50 %, но в задаче классификации по трем (и более) классам выбирается класс с наибольшей вероятностью/достоверностью в качестве результата прогнозирования классификатора.

characteristic – ROC) и вычисление площади под ROC-кривой (area under ROC curve – AUC). ROC-кривая отображает коэффициент ложноположительных случаев (FP/(FP+TN)) по оси x и коэффициент истинно положительных случаев (TP/(TP+FN), также обозначаемый термином «полнота» (recall)) по оси y. Каждая точка на кривой соответствует пороговой оценке и представляет пару (FPR, TPR) для этой пороговой оценки. Площадь под ROC-кривой можно интерпретировать как вероятность того, что случайно выбранный положительный элемент имеет более высокую оценку, чем случайно выбранный отрицательный элемент. С учетом этой интерпретации легко понять, что наихудшим случаем является AUC 0,5, т. е. равнозначность случайного выбора порядка элементов выборки[1].

На рис. 2.10 показан пример ROC-кривой и линии y = x, нанесенной на график для сравнения. Поскольку область под ROC-кривой весьма велика, мы воспользовались логарифмическим масштабом (шкалой) для увеличения левой части графика, где хорошо будут видны различия между высокоэффективными моделями. Если рассматривать только область кривой, показанную с высокой точностью, то может потребоваться вычисление порогового значения для коэффициента ложноположительных случаев, как, например, 1 %.

Полезным свойством области под ROC-кривой является то, что на нее не влияет отклонение (смещение) выборки. Таким образом, если в выборке имеются два класса с различными весами, то площадь под ROC-кривой, полученной для окончательного результативного набора данных, будет репрезентативной и для AUC на неотобранном наборе данных.

Общепринятой метрикой, устанавливающей ограничения действительной полезности, является F-оценка, или F-мера, определяемая по следующей формуле:

$$F_{\alpha} = \frac{1+\alpha}{\frac{1}{\text{precision}} + \frac{\alpha}{\text{recall}}}.$$

F-оценка объединяет точность и полноту и жестко штрафует экстремальные значения. Тем не менее при этом требуется выбор порогового значения и относительных весовых коэффициентов для точности и полноты (параметризованных с помощью α).

Кластеризация

Беда никогда не приходит одна. Например, если кто-то пытается проникнуть в вашу сеть извне, то существует высокая вероятность того, что такие попытки будут предприниматься многократно, до тех пор пока злоумышленники действительно не прорвутся в сеть. Или если некто рассылает фармацевтический спам, то распространяться будет огромное количество писем, чтобы достаточное число людей оказалось буквально под завалами этого мусора. Таким образом, ваша работа в качестве службы защиты будет выполняться проще, если удастся разделить трафик на группы по действующим источникам, а затем блокировать

[1] Если площадь под ROC-кривой (AUC) меньше 0,5, то с помощью реверсирования меток классификатора можно получить классификатор с AUC > 0,5.

трафик от злоумышленников. Такой процесс сегментации называют кластеризацией (clastering).

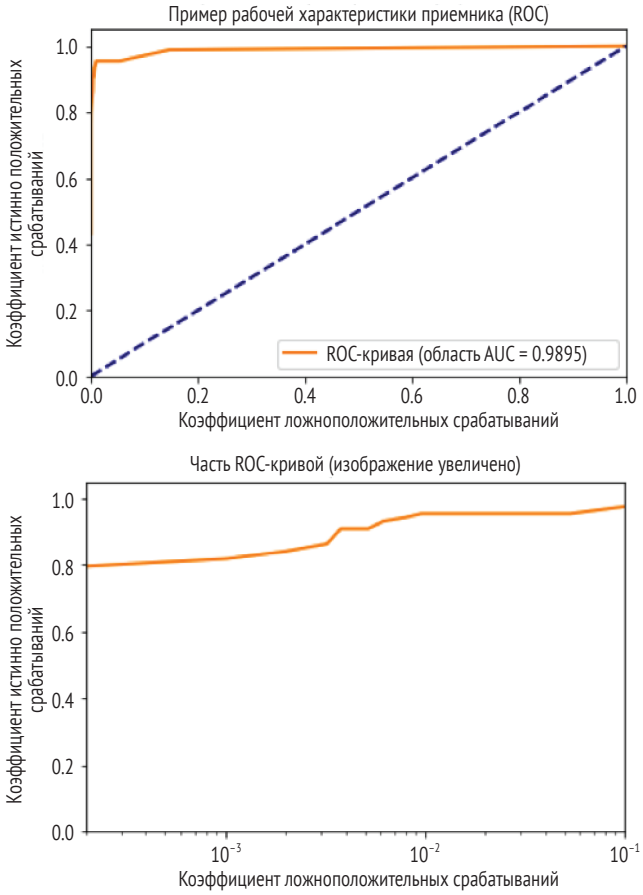


Рис. 2.10 ❖ ROC-кривые

В этом разделе кратко рассматриваются некоторые общие методики кластеризации данных. Разумеется, группировка конкретных данных сама по себе не является целью, конечная цель – определение кластеров, которые содержат признаки вредоносной деятельности. Поэтому будут также рассматриваться разнообразные методики присваивания меток кластерам, сгенерированным различными алгоритмами.

Алгоритмы кластеризации

Геометрическое мысленное представление о кластеризации вполне очевидно: необходимо сгруппировать точки данных, которые «расположены близко друг к другу» в определенном смысле. Таким образом, для работы любого алгоритма необходимо получить некоторый конкретный способ измерения «близости» точек. Такая характеристика называется метрикой (metric). Устанавливаемая мет-

рика и используемый алгоритм кластеризации зависят от формы представления данных. Например, данные могут состоять из векторов действительных чисел, списков элементов или последовательностей битов. Рассмотрим наиболее часто используемые алгоритмы.

Группировка

Самый элементарный метод кластеризации настолько прост, что обычно его даже не считают методом кластеризации: выбор одного или нескольких измерений и определение каждого кластера как набора элементов, которые имеют одинаковые значения в этом измерении. В синтаксисе языка SQL для этого предназначен оператор `GROUP BY`, поэтому такую методику называют группировкой (grouping). Например, если выполняется группировка по IP-адресу, то определяется по одному кластеру для каждого IP-адреса, а элементами такого кластера будут объекты, совместно использующие один и тот же IP-адрес.

В начале главы мы уже наблюдали работу по методике группировки, когда рассматривали сверхинтенсивные запросы, приходящие на один IP-адрес. Такой подход равнозначен кластеризации по IP-адресу, при этом помечаются как вредоносные все кластеры с более чем 20 запросами в секунду. Этот пример демонстрировал достаточную мощь метода кластеризации даже при простой группировке. В дальнейшем вы убедитесь, что с помощью подобной методики можно сделать достаточно много, не обращаясь к более сложным алгоритмам.

Метод k-средних

Метод k-средних – это обычно самый первый алгоритм, который вспоминается при необходимости выполнения кластеризации. Метод k-средних применяется к векторам действительных значений, когда известно предполагаемое число кластеров, которое обозначается буквой k. Цель алгоритма – определить каждую точку данных как принадлежащую определенному кластеру, при этом сумма расстояний от каждой точки до центра своего кластера минимизируется. Ниже приведена формула для вычисления расстояния, которое является обычным эвклидовым расстоянием в векторном пространстве:

$$d(x, y) = \sqrt{\sum_i (x_i - y_i)^2}.$$

В математическом смысле алгоритм k-средних вычисляет такое распределение точек по кластерам $f: X \to \{1, \ldots, k\}$, которое минимизирует функцию потерь:

$$L(X) = \sum_i d(x_i, c_{f(x_i)}),$$

где $X = \{x_1, \ldots, x_n\}$ – исследуемый набор данных, c_j – центр (центроид) j-го кластера, d – расстояние между двумя точками. Значение L(X) называется «инерцией» (inertia).

Стандартный алгоритм вычислений по методу k-средних приведен ниже.

1. Выбор k центров (центроидов) $c_1, \ldots, c_k$ случайным образом.
2. Определение каждой точки данных x_i как принадлежащей ближайшему центру (центроиду).
3. Повторное вычисление центроидов c_j с учетом среднего значения всех точек данных, принадлежащих j-му кластеру.

4. Повторение шагов 2 и 3 до полной сходимости алгоритма, т. е. до того момента, когда разность между значениями L(X) в последовательных итерациях становится меньше предварительно определенного порогового значения.

Метод k-средних представляет собой простой и эффективный алгоритм кластеризации, который хорошо масштабируется для весьма больших наборов данных. Но при использовании этого алгоритма необходимо учитывать некоторые присущие ему проблемы и особенности:

- поскольку k является фиксированным параметром алгоритма, необходимо правильно выбирать его значение. Если известно, сколько кластеров исследуется (например, при попытке кластеризации различных групп вредоносных программ), можно просто выбрать значение k равным этому числу. В других случаях придется экспериментировать с различными значениями k. Кроме того, часто выбирают значения k из диапазона от числа классов (меток) данных до утроенного этого же числа, если категории прерывающиеся. Внимание: функции потерь, вычисляемые с различными значениями k, нельзя сравнивать между собой;
- перед использованием метода k-средних необходимо обязательно нормализовать обрабатываемые данные. Обычная нормализация представляет собой отображение j-й координаты x_{ij} в $(x_{ij}-\mu_j)/\sigma_j$, где μ_j – меана j-х координат, а σ_j – стандартное отклонение.
  Чтобы убедиться в необходимости нормализации, рассмотрим двумерный набор данных, первая координата которого находится в диапазоне между 0 и 1, а вторая координата – в диапазоне от 0 до 100. Очевидно, что вторая координата будет в значительно большей степени воздействовать на функцию потерь, поэтому информация о том, насколько близки точки по первой координате, вероятнее всего, будет потеряна;
- не используйте метод k-средних с категориальными признаками, даже если их можно представить в числовом виде. Например, можно кодировать цвета «красный», «зеленый», «синий» как 0, 1, 2 соответственно, но эти числа не имеют смысла в векторном пространстве – нет никаких оснований считать, что синий цвет в два раза дальше от красного, чем зеленый. Эту проблему можно разрешить с помощью унитарного, или прямого, кодирования категориальных признаков как нескольких бинарных признаков (как было показано в рабочем примере в предыдущей части этой главы), но при этом необходимо решить следующую проблему;
- следует избегать применения метода k-средних с бинарными признаками. Метод k-средних иногда можно использовать с бинарными признаками, кодирующими два варианта ответов как 0 и 1 или –1 и 1, но результаты могут быть непредсказуемыми. Бинарный признак может стать доминирующим признаком, определяющим кластер, или его информация может быть полностью потеряна;
- метод k-средних снижает эффективность при большом числе измерений из-за «проклятия размерности» – все точки приблизительно равноудалены друг от друга. Для получения наилучших результатов используйте метод k-средних при небольших размерностях или после применения алгоритма снижения размерности, такого как метод анализа главных компонентов

(principal component analysis – PCA). Другой вариант – использование расстояния Чебышёва, или L_∞-расстояния, когда расстояние между двумя точками принимается как максимальная разность по любой координате:

$$d(x,y) = \max_i(|x_i - y_i|);$$

- метод k-средних работает наилучшим образом при случайном выборе начальных центроидов. Но такой выбор может затруднить повторное воспроизведение результатов. Рекомендуется пробовать различные варианты выбора начальных центроидов, для того чтобы увидеть, как результаты зависят от инициализации;
- при использовании метода k-средних предполагается, что кластеры по своей природе являются сферическими (шаровидными). Поэтому можно сделать вывод, что метод k-средних неудовлетворительно работает в несферических распределениях, подобных показанному на рис. 2.11.

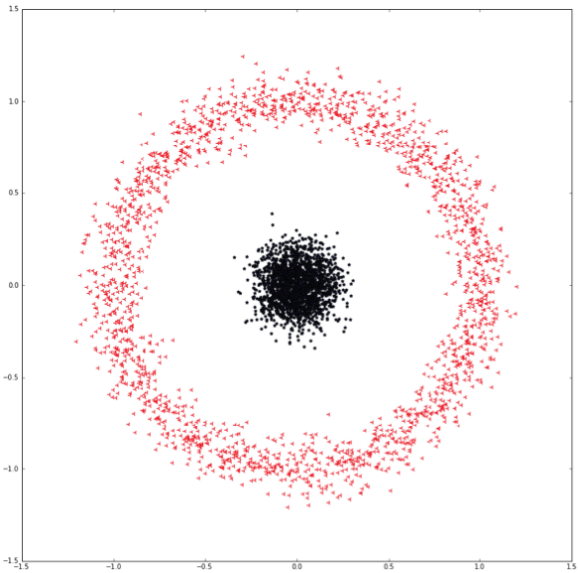


Рис. 2.11 ❖ Несферическое распределение данных

Иерархическая кластеризация

В отличие от алгоритма k-средних, методы иерархической кластеризации не параметризованы по значению k, выбираемому оператором (пользователем), т. е. по числу кластеров, которые вы намерены создать. Выбор правильного значения k – нестандартная задача, решение которой может существенно повлиять на результаты кластеризации. Метод агломеративной (снизу вверх) иерархической кластеризации (agglomerative (bottom-up) hierarchical clustering) создает кластеры по следующей схеме (рис. 2.12).

1. Каждая точка данных помещается в свой собственный кластер (рис. 2.12, самый нижний уровень).
2. Объединяются два кластера, похожие друг на друга в наибольшей степени, при этом «наибольшая похожесть» определяется по метрике расстояния,

например как эвклидово расстояние, или расстояние Махаланобиса (Mahalanobis distance).

3. Шаг 2 повторяется до тех пор, пока не останется только один кластер (рис. 2.12, самый верхний уровень).
4. Выполняется проход по уровням получившегося дерева или дендрограммы (dendrogram), и выбирается уровень, который дает наиболее точный результат кластеризации.

Дивизивная, или дивизионная (сверху вниз), иерархическая кластеризация (divisive (top-down) hierarchical clustering) представляет другую форму иерархической кластеризации, работающую в противоположном направлении. Вместо того чтобы начать с количества кластеров, равного числу точек данных, мы начинаем с единственного кластера, содержащего все точки данных, и выполняем разделение кластеров на основе метрики расстояния. Процесс останавливается, когда каждой точке данных соответствует собственный отдельный кластер.

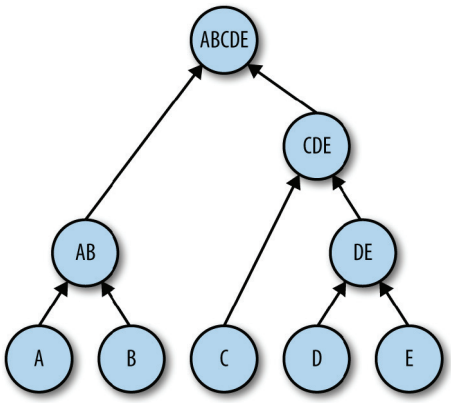


Рис. 2.12 ❖ Дендрограмма
метода агломеративной иерархической кластеризации

При практическом использовании иерархической кластеризации необходимо помнить о следующих важных особенностях этих методов:

- иерархическая кластеризация генерирует модель дерева, или дендрограмму, как показано на рис. 2.12. Эта модель может быть более сложной для анализа и потребовать большего объема памяти для хранения, чем центроиды, сгенерированные методом k-средних, но при этом сообщает больше информации о внутренней структуре данных. Если самым главным критерием является компактность модели или простота анализа, то иерархическая кластеризация – не самый лучший выбор;
- метод k-средних работает лишь с небольшим набором метрик расстояния (главным образом используется эвклидово расстояние) и требует для работы числовые данные. Методы иерархической кластеризации работают практически со всеми типами метрик расстояния или функций сходства, а кроме того, генерируют результаты, которые можно сравнивать в числовом выражении (например, C больше похоже на A, чем на B). Этот метод можно применять к категориальным данным, к смешанным типам данных,

к строкам, изображениям и т. п. при условии, что предоставлена соответствующая функция расстояния;

- методы иерархической кластеризации имеют высокую сложность по времени, поэтому не могут считаться подходящими для больших наборов данных. Если обозначить через n число точек данных, то для агломеративной иерархической кластеризации сложность по времени составляет $O(n^2 \log(n))$, а для наивной дивизивной кластеризации сложность по времени выражается как $O(2^n)$.

Чувствительное к местоположению хеширование

Метод k-средних хорошо работает для задач по определению элементов, наиболее близких друг к другу, при этом каждый элемент может быть представлен в виде последовательности чисел (т. е. в виде вектора в векторном пространстве). Но для многих элементов, подлежащих кластеризации, не так-то просто подобрать соответствующее числовое представление. Классическим примером являются текстовые документы, которые имеют различную длину (размер в символах) и отличаются бесконечным разнообразием используемых слов и порядка слов. Другой пример – списки, например набор IP-адресов, доступных конкретному пользователю, или набор всех друзей некоторого пользователя в социальном графе.

Наиболее часто используемой метрикой сходства для неупорядоченных наборов данных является коэффициент, или мера сходства, Жаккара (Jaccard similarity). Мера сходства Жаккара определяется как отношение числа элементов, общих для двух наборов данных, к общему числу всех элементов в обоих наборах. Более точно мера сходства Жаккара для двух наборов данных X и Y определяется следующей формулой:

$$J(X,Y) = \frac{|X \cap Y|}{|X \cup Y|}.$$

Для генерации кластеров в тех случаях, когда элементы являются наборами (множествами) данных, необходимо только найти группы элементов, для которых мера сходства Жаккара очень высока. Проблема здесь заключается в том, что сложность вычислений возрастает в квадратичной зависимости от количества элементов, поэтому при увеличении наборов данных поиск кластеров быстро становится практически невозможным. Чувствительное к местоположению хеширование (locality-sensitive hashing – LSH) пытается решить эту проблему. Обычно LSH не рассматривается как алгоритм кластеризации, но его можно использовать как метод группировки схожих элементов по некоторому представлению «расстояния». При этом достаточно успешно достигается тот же эффект, что и при использовании более привычных алгоритмов кластеризации.

Если элементы, подлежащие кластеризации, представляют собой неупорядоченные наборы данных (например, любой текстовый документ), то первым шагом становится преобразование их в упорядоченное множество. Для текстовых документов самым простым преобразованием является формирование мешка слов (bag of words), т. е. простого списка слов, содержащихся в документе. В зависимости от конкретной реализации повторяющиеся слова могут включаться или не включаться многократно в этот список, а кроме того, могут исключаться шумовые слова, такие как артикли «a», «the» или предлоги типа «of».

Но при преобразовании в мешок слов теряется важное свойство текстового документа – порядок слов. Для сохранения информации о порядке слов преобразование обобщается в форме «черепицы» (shingling): составляется список (перекрывающихся) последовательностей слов, расположенных рядом в документе. Например, если рассматривать документ «the quick brown fox jumps over the lazy dog», то «черепицы» из трех последовательных слов должны выглядеть следующим образом:

{(the, quick, brown), (quick, brown, fox), (brown, fox, jumps), (fox, jumps, over), (jumps, over, the), (over, the lazy), (the, lazy, dog)}

Также можно выполнить преобразование в «черепицу» на уровне символов – это может оказаться полезным для коротких документов или текстовых строк, которые неудобно разбирать по словам.

Теперь при рассмотрении набора данных, состоящего из неупорядоченных множеств, возникает вопрос: как эффективно определить элементы, похожие друг на друга, в соответствии с мерой сходства Жаккара. Первый шаг – преобразование каждого множества в короткую «сигнатуру» так, чтобы похожие документы имели похожие сигнатуры. Стандартным алгоритмом для такой задачи является MinHash, который работает по следующей схеме.

1. Выбирается k независимых хеш-функций h_i, которые принимают на входе произвольные данные и генерируют на выходе 32-битовые целые значения.
2. Берется элемент $x = \{x_1, \ldots, x_n\}$, для каждого i определяется $m_i(x) = \min(\{h_i(x_i), \ldots, h_i(x_n)\})$.
3. В результате формируется сигнатура $H(x) = (m_1(x), \ldots, m_k(x))$.

Главное свойство этого алгоритма: если поведение выбранных хеш-функций достаточно случайно, то для двух элементов x и y вероятность того, что $m_i(x) = m_i(y)$, равна мере сходства Жаккара для x и y[1]. Поскольку используется k независимых хеш-функций, можно оценить J(x,y) с помощью простого подсчета числа коллизий $m_i(x) = m_i(y)$ и деления на k. При увеличении числа k эта оценка становится более точной.

После завершения вычисления сигнатур MinHash применяется только что рассмотренное главное свойство, т. е. предполагается, что два элемента со значительно перекрывающимися сигнатурами обладают высокими коэффициентами сходства Жаккара. Таким образом, для создания кластеров схожих элементов необходимо найти группы сигнатур, которые перекрываются по многим позициям (знакам числа). Для выполнения этой задачи существует два известных способа:

- для каждого i вычисляется обратное отображение из $m_i(x)$ во все элементы x, которым соответствует это конкретное хеш-значение. Теперь для задан-

[1] Попробуем понять, почему возникает это свойство. Сначала можно ввести упрощение, предположив, что между хеш-функциями не возникают коллизии, так что все функции $h_j(x_j)$ отличаются друг от друга, следовательно, при $m_i(x) = m_i(y)$ означает, что существуют некоторые значения j, j′ такие, что $x_j = y_{j'}$, или, другими словами, что минимальное хеш-значение соответствует элементу, общему для двух исследуемых наборов данных. Теперь рассмотрим все элементы объединения x ∪ y как множество $\{z_1, \ldots, z_t\}$. Если хеш-функция h_i рассматривается как случайная, то вероятность того, что наименьшее значение $h_i(z_j)$ находится в пересечении x ∩ y, в точности равно отношению $|x \cap y|/|x \cup y|$, которое и является мерой сходства Жаккара J(x, y).

ного элемента x_0 можно найти все значения x, которые соответствуют $m_i(x_0)$, и сохранить значения, совпадающие, как минимум, по t знакам числа;

- выполняется преобразование k хеш-значений в b блоков, при этом каждый блок состоит из r «рядов». Пусть L – вторая хеш-функция, которая отображает хеш-значения ряда в строку фиксированного размера. Можно сгруппировать элементы, хеш-значения L которых совпадают по всем рядам, как минимум, в одном блоке. Точная настройка значений b и r позволяет найти компромисс между ложноположительными случаями (т. е. различные элементы при совпадении L) и ложноотрицательными случаями (т. е. схожие элементы при несовпадении L): увеличение b снижает количество ложноотрицательных случаев, а увеличение r снижает количество ложноположительных случаев. Более подробно это описано в главе 3 книги Mining of Massive Datasets, 2nd ed., by Jure Leskovek, Anand Rajaraman and Jeffrey D. Ullman (издательство Cambridge University Press).

MinHash – это всего лишь один пример чувствительной к местоположению функции хеширования, т. е. функции, устанавливающей соответствие между элементами, близкими друг к другу в пространстве входных данных, и значениями, близкими друг к другу в пространстве выходных данных. Если «близость» измеряется какой-либо метрикой, отличающейся от меры сходства Жаккара, например эвклидовым расстоянием или расстоянием Хэмминга (Hamming distance), то существуют специализированные функции, которыми можно воспользоваться вместо MinHash в описанной выше процедуре. Эти функции подробно описаны в книге Лесковека (Leskovek) и др.

k-мерные деревья

k-мерное дерево, или просто k-d дерево (k-d tree), – это бинарное дерево, которое хранит данные в формате, оптимизированном для многомерного пространственного анализа. Формирование k-d дерева можно рассматривать как предварительный этап алгоритмов классификации по методу k-ближайших соседей (k-NN), но эту методику также можно считать самостоятельным алгоритмом кластеризации. Как и при иерархической кластеризации, алгоритм создает структуру дерева, но здесь кластеры хранятся в листьях, а не в промежуточных узлах.

Типовой алгоритм формирования k-d деревьев приведен ниже. Для каждого узла, отличающегося от листа, выполняются следующие действия.

1. Выбор размерности для разделения.
2. Выбор точки разделения (например, медианы этой размерности в подпространстве признаков данного узла).
3. Разделение подпространства в соответствии с выбранной размерностью и точкой разделения.
4. Прекращение разделения подпространств, когда текущее подпространство содержит меньше элементов, чем определенное количество элементов выборки для отдельного подпространства, leaf_size (например, если leaf_size == 1, то каждый узел-лист в дереве должен представлять подпространство признаков, содержащее только один элемент выборки).

Результатом этой процедуры является дерево бинарного поиска в подпространствах признаков, при этом объединение всех подпространств узлов-листов формирует полное пространство признаков. При создании моделей k-d деревьев

для поиска ближайших соседей бинарное дерево с разделенным пространством должно быть сохранено как дополнение к точкам тренировочных данных. Более того, дополнительные данные о том, какие элементы выборки принадлежат конкретным узлам-листам, также должны быть сохранены в модели. Таким образом, для хранения подобной модели требуется еще большее пространство, что делает ее менее экономной (с точки зрения потребления ресурса памяти), чем оригинальные модели k-NN.

k-d деревья можно использовать для ускорения поиска ближайших соседей (например, в процессе k-NN классификации). При поиске k ближайших соседей элемента выборки x выполняются следующие действия.

1. Процедура начинается с корня дерева, выполняется проход по дереву в поисках узла, представляющего подпространство признаков, которое содержит x.
2. Анализ подпространства признаков, содержащего x:
   – если leaf_size == k, то в качестве результата возвращаются все точки этого подпространства;
   – если leaf_size > k, то выполняется полный исчерпывающий (методом грубой силы или простого перебора) поиск в этом подпространстве признаков для k точек, самых близких к x, и, как результат, возвращаются найденные k точек;
   – если leaf_size < k, то сохраняются все точки в этом подпространстве как часть результата, и процедура продолжается (переход к следующему шагу).
3. Выполняется следующий шаг по дереву и анализ подпространства признаков, представленного в текущем узле. Продолжается добавление соседних точек к результату до тех пор, пока все k соседей не будут найдены. Этот шаг повторяется столько раз, сколько необходимо для получения k точек.

Как и алгоритм k-NN, k-d деревья обычно не подходят для данных с высокой размерностью[1], они даже более громоздки, чем модели k-NN. Тем не менее k-d деревья очень удобны для быстрого поиска ближайших соседей со сложностью по времени в среднем O(log n). Варианты k-d деревьев разработаны для решения разнообразных задач с помощью этого алгоритма. Один из примеров – деревья квадрантов, или квадродеревья (quadtrees), оптимизированные для поиска в двумерных пространствах.

DBSCAN

Основанная на плотности пространственная кластеризация для приложений с шумами (Density-Based Spatial Clustering of Applications with Noise – DBSCAN)[2] – один из самых известных и широко применяемых алгоритмов кластеризации, благодаря своей способности показывать высокую эффективность в различных ситуациях. В отличие от метода k-средних, количество кластеров не определяется предварительно оператором, а выводится из данных. В отличие от иерархической

---

[1] Для пространств признаков с высокой размерностью эффективность работы k-d деревьев сравнима с эффективностью линейного поиска простым перебором (методом грубой силы).

[2] *Ram Anant, et al.* A Density Based Algorithm for Discovering Density Varied Clusters in Large Spatial Databases. International Journal of Computer Applications 3:6 (2010).

кластеризации, основанной на метрике расстояний, DBSCAN представляет собой основанный на плотности алгоритм, который разделяет наборы данных на подгруппы в областях высокой плотности. Рассмотрим некоторые термины и параметры, используемые этим алгоритмом:

- в алгоритм передаются два параметра:
  - ε определяет радиус относительно конкретной точки, внутри которого выполняется поиск соседей;
  - minPoints – минимальное количество точек, требуемое для формирования кластера;
- каждая точка данных классифицируется как основная точка, краевая точка или шумовая точка:
  - основные точки (core points) – точки, которые имеют, как минимум, minPoints точек (соседей) внутри своего ε-радиуса;
  - краевые точки (border points) сами по себе не являются основными точками, но находятся внутри ε-радиуса какой-либо основной точки;
  - шумовые точки (noise points) не являются ни основными, ни краевыми точками.

В простейших реализациях этот этап классификации выполняется с помощью итерации по всем точкам набора данных с вычислением их расстояния до всех прочих точек в наборе. Затем каждая точка связывается со своими соседями (с точками, находящимися на расстоянии, меньшем радиуса ε). Используя эту информацию, можно пометить все точки как основные, краевые или шумовые точки. После классификации все точки данных в исследуемом наборе принадлежат к одному из перечисленных типов точек, и алгоритм DBSCAN продолжает работу следующим образом.

1. Случайный выбор точки P из всех непосещенных (unvisited) точек.
2. Если P – не основная точка, то пометить ее как посещенную (visited) и продолжить.
3. Если P – основная точка, то сформировать вокруг нее кластер, выполнить рекурсивный поиск и захватить все другие точки в пределах ε-радиуса точки P, а также все прочие точки, находящиеся в пределах ε-радиуса всех основных точек, захваченных этим кластером.

   Предположим, что существует некоторая основная точка Q в пределах ε-радиуса точки P. Q (вместе со всеми ее краевыми точками) будет добавлена в кластер, сформированный вокруг точки P. Если существует другая основная точка R внутри ε-радиуса точки Q, то такая точка R (вместе со всеми ее краевыми точками) также будет добавлена в кластер, сформированный вокруг точки P.

   Этот рекурсивный шаг повторяется до тех пор, пока не останется основных точек, которые могут быть захвачены.
4. Процедура повторяется до тех пор, пока все точки в наборе данных не будут помечены как посещенные.

Несмотря на то что алгоритм DBSCAN демонстрирует очень хорошую работу в разнообразных ситуациях, он не лишен некоторых недостатков:

- алгоритм DBSCAN неэффективен, когда кластеры в наборе данных имеют различные плотности. Это затрудняет выбор значений ε и minPoints, оптимальных для всех кластеров в наборе данных. Упорядочение точек для об-

наружения кластерной структуры (Ordering Points to Identify the Clustering Structure – OPTICS)[1] – очень похожий на DBSCAN алгоритм, который устраняет этот недостаток, вводя пространственное упорядочение в процесс выбора точек за счет снижения скорости работы;

- правильный выбор параметров ε и minPoints очень важен для высокой эффективности алгоритма. Выбор правильных значений для этих параметров может быть весьма сложной задачей, если у вас нет полного понимания распределения и плотностей данных в исследуемом наборе;
- алгоритм DBSCAN является недетерминированным и может давать различные результаты в зависимости от выбора точек, которые будут посещаться в первую очередь на шаге 1 процедуры случайного выбора;
- алгоритм DBSCAN показывает слабую эффективность для наборов данных с высокой размерностью, так как в основном использует эвклидово расстояние как метрику расстояния, поэтому подвержен «проклятию размерности»;
- если исследуемый набор данных представляет собой выборку из необработанного («сырого») источника, то используемый метод выборки может существенно изменить характеристики плотности данных. В таких случаях алгоритмы, основанные на плотности, не применимы, так как выдаваемые ими результаты полагаются на точное представление реальных характеристик плотности данных.

Оценка результатов кластеризации

Иногда трудно понять смысл результатов операций кластеризации. Оценка алгоритмов обучения с учителем является более простой задачей, поскольку мы имеем доступ к действительно истинным меткам: можно просто подсчитать количество элементов выборки, которым алгоритм присваивает метки «правильно» и «неправильно». При обучении без учителя, вероятнее всего, доступа к меткам не будет, тем не менее процедура оценки становится намного более простой[2]. Например, часто используемыми метриками для оценки результатов кластеризации при использовании реально истинных меток являются:

- однородность, или гомогенность (homogeneity), – степень, в которой каждый кластер содержит только члены единственного класса;
- полнота (completeness) – степень принадлежности всех членов определенного класса одному и тому же кластеру.

Среднее гармоническое этих двух метрик известно как V-мера (V-measure)[3] – оценка, основанная на энтропии, представляющая точность операции кластеризации и вычисляемая по формуле:

---

[1] *Mihael Ankerst, et al.* OPTICS: Ordering Points to Identify the Clustering Structure. SIGMOD Record 28:2 (1999): 49–60.

[2] *W.M. Rand*. Objective Criteria for the Evaluation of Clustering Methods. Journal of the American Statistical Association 66 (1971): 846–850.

Nguyen Xuan Vinh, Julien Epps and James Bailey. Information Theoretic Measures for Clustering Comparison: Is a Correction for Chance Necessary? Proceedings of the 26th Annual International Conference on Machine Learning (2009): 1073–1080.

[3] *Andrew Rosenberg and Julia Hirschberg*. V-Measure: A Conditional Entropy-Based External Cluster Evaluation Measure. Proceedings of the 2007 Joint Conference on Empirical Methods in Natural Language Processing and Computational Natural Language Learning (2007): 410–420.

$$v = \frac{2hc}{h+c},$$

где h – однородность, c – полнота.

Мы будем пользоваться реализациями этих метрик оценки кластеризации библиотеки scikit-learn (**https://scikit-learn.org/stable/modules/clustering.html#homogeneity-completeness-and-v-measure**) в главе 5.

С другой стороны, тот факт, что мы рассматриваем кластеризацию вообще, означает, что, вероятнее всего, у нас не будет доступа к каким-либо реально истинным меткам для исследуемых наборов данных. Мы можем больше не пользоваться V-мерой, поскольку однородность и полнота могут быть измерены только при сравнении прогнозируемых меток с реально истинными метками. В этом случае мы должны полагаться на сигналы, сгенерированные напрямую из самой тренировочной модели. Операция кластеризации считается успешной, если все элементы выборки, распределенные в один и тот же кластер, похожи друг на друга, а элементы выборки, разнесенные по разным кластерам, полностью отличаются друг от друга. Для измерения этих характеристик существуют два широко известных метода:

- силуэтный коэффициент (silhouette coefficient): эта оценка вычисляется отдельно для каждого элемента выборки в наборе данных. С использованием одной из метрик расстояния (например, эвклидово расстояние) вычисляется два средних расстояния для некоторого элемента выборки x:
  - a – среднее расстояние между элементом выборки x и всеми прочими элементами выборки в одном и том же кластере;
  - b – среднее расстояние между элементом выборки x и всеми прочими элементами выборки в следующем ближайшем кластере.

  Затем силуэтный коэффициент s определяется по следующей формуле[1]:

  $$s = \frac{b-a}{\max(a,b)}.$$

  Для результата кластеризации абсолютно неправильным будет состояние, при котором a намного больше b, т. е. числитель становится отрицательным. Поскольку знаменатель никогда не будет отрицательным (метрика расстояния не принимает отрицательных значений), коэффициент s является отрицательным числом. В действительности значение s ограничено интервалом от –1 до +1. Чем ближе s к +1, тем лучше результат кластеризации. Когда s очень близок к 0, кластеры являются сильно перекрывающимися. Тем не менее обнаружено, что силуэтный коэффициент в большей степени подходит для алгоритмов кластеризации на основе расстояния (например, для метода k-средних), на основе сеточных вычислений (например, STING[2]), а также для иерархических алгоритмов кластеризации. Силуэтный

[1] *Peter Rousseeuw*. Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis. Journal of Computational and Applied Mathematics 20 (1987): 53–65.

[2] *Wei Wang, Jiong Yang and Richard Muntz*. STING: A Statistical Information Grid Approach to Spatial Data Mining. Proceedings of the 23rd International Conference on Very Large Data Bases (1997): 186–195.

коэффициент малопригоден для кластеров, формируемых по алгоритмам на основе плотности, таких как DBSCAN и OPTICS, потому что ни a, ни b не учитывают плотности кластера. Кроме того, поскольку силуэтный коэффициент непременно должен вычисляться независимо для каждого элемента выборки в исследуемом наборе данных, эта процедура может оказаться достаточно медленной;

- индекс Калински–Харабаша (Calinski–Harabasz (C–H) index)[1] выше (лучше), когда кластеры плотные и четко разделены. Этот индекс эффективности кластеризации в большей степени близок к тому, как люди оценивают результаты кластеризации. Результат кластеризации считается хорошим, если созданы визуально разделяемые и плотно упакованные группы элементов. C–H-индекс использует две меры:
  - W_k – дисперсия внутри кластера (within-cluster dispersion), матрица расстояний между элементами в кластере и геометрическим центром кластера;
  - B_k – дисперсия между группами (between-group dispersion), матрица расстояний между центром кластера и центрами всех прочих кластеров (здесь k – количество кластеров в модели).

  Оценка (индекс) C–H вычисляет отношение между W_k и B_k (где N – количество элементов в наборе данных, а tr – след матрицы[2]):

$$s = \frac{\mathrm{tr}(B_k)}{\mathrm{tr}(W_k)} \times \frac{N-k}{k-1}.$$

  Индекс C–H может вычисляться гораздо более эффективно, чем силуэтный коэффициент. Даже притом что индекс C–H остается пригодным в большей степени для кластеров на основе плотности, он сравнительно более надежен, чем силуэтный коэффициент.

Вне зависимости от того, какой метод выбран, неизбежно будут существовать присущие ему ограничения в оценке результатов кластеризации без каких-либо меток, присвоенных человеком. Если есть подозрение, что модели кластеризации на практике работают недостаточно хорошо, несмотря на правильно выбранные оценочные метрики, возможно, потребуются дополнительные ресурсы для пометки подмножества элементов вручную и использование методик обучения с частичным привлечением учителя для оценки такой модели.

Резюме

Машинное обучение в своей основе является процессом использования хронологически накопленных данных для вывода алгоритма прогнозирования для новых, ранее не встречавшихся данных. В этой главе главное внимание было уделено классификации (т. е. определению того, к какой категории принадлежит каждая точка данных) и кластеризации (т. е. определению того, какие точки данных по-

[1] *Tadeusz Caliński and J. A. Harabasz*. A Dendrite Method for Cluster Analysis: Communications in Statistics. Theory and Methods 3 (1974): 1–27.

[2] След матрицы (trace of a matrix) определяется как сумма элементов ее диагонали (от верхнего левого до нижнего правого).

хожи друг на друга). Классификация может выполняться с учителем, в этом случае ранее накопленные данные имеют метки класса, или без учителя, тогда ранее накопленные данные меток не имеют (или метки существуют только для весьма незначительной части данных). Последний случай более подходит для кластеризации.

Мы кратко, без подробностей рассмотрели ряд наиболее широко применяющихся алгоритмов классификации и кластеризации, а также определили, для каких задач наилучшим образом подходит каждый алгоритм. Для подробного описания всех рассмотренных здесь алгоритмов может потребоваться отдельная книга. Такая книга существует: Хасти (Hastie), Тибширани (Tibshirani) и Фридман (Friedman) подробно рассматривают теорию и практику в своей книге «The Elements of Statistical Learning» (издательство Springer), и мы рекомендуем ее всем интересующимся.

Обсуждаемые здесь алгоритмы для эффективной работы требуют значительного объема данных – чем больше данных, тем лучше. Но как быть, если событие, которое вы пытаетесь определить, происходит чрезвычайно редко? Решение этой задачи является темой следующей главы.

Глава 3

Выявление аномалий

В этой главе рассматриваются способы выявления непредвиденных событий, или аномалий (anomalies), в системах. С точки зрения обеспечения безопасности сетей и хостов выявление аномалий (anomaly detection) означает определение случаев непредусмотренных вторжений или ранее не замеченных уязвимостей. В среднем выявление уязвимости в системе требует около десяти дней (**http://www.verizonenterprise.com/verizon-insights-lab/dbir/2016/**). Но после того как атакующий получил доступ к системе, вред обычно наносится в течение нескольких дней или даже быстрее. Вне зависимости от сущности атаки – «утечка» данных, вымогательство с использованием программ-шантажистов, навязывание рекламы или целевая кибератака («развитая устойчивая угроза», APT) – очевидно, что время не на стороне защищающихся.

Важность выявления аномалий не ограничена сферой обеспечения безопасности. Если говорить более обобщенно, то выявление аномалий – это любой метод поиска событий, которые не соответствуют ожиданиям и предположениям. В системах, для которых надежность чрезвычайно важна, можно воспользоваться методами выявления аномалий для обнаружения ранних признаков аварийной ситуации, заставляющих операторов провести тщательное превентивное исследование. Например, если компания, занимающаяся обеспечением потребителей электроэнергией, может обнаружить аномалии в своей электросети и устранить их, это позволит избежать крупных издержек и потерь как следствия отказов других компонентов системы из-за резких скачков напряжения. Другим важным практическим применением методов выявления аномалий является сфера разоблачения мошенничеств. Мошенничество в области финансов часто может быть выявлено среди огромного количества законных транзакций при помощи изучения шаблонов обычных событий и обнаружения встречающихся отклонений от них.

Терминология

В этой главе термины «промах» (outlier) и «аномалия» (anomaly) используются как взаимозаменяемые. С другой стороны, существует важное различие между выявлением промахов (в статистике их также называют выбросами) и выявлением новизны (novelty detection). Задача выявления новизны предполагает обучение на представлении «обычных» данных при использовании для этого данных, не содержащих никаких промахов (выбросов). При решении задачи выявления промахов обучение проводится на данных, содержащих и обычные данные, и промахи. Важность этого

различия обсуждается ниже в текущей главе. Выявление новизны и выявление промахов представляют собой формы выявления аномалий.
Неаномальные точки данных мы называем обычными (regular) данными. Не следует путать их с обозначениями нормальных (normal), или стандартных (standard), данных. В этой главе термин «нормальный» используется в том же смысле, что и в математической статистике, т. е. соответствующий нормальному распределению, или распределению Гаусса. Термин «стандартный» также используется в статистическом смысле, т. е. обозначает нормальное распределение с математическим ожиданием, равным нулю, и стандартным отклонением, равным 1.

Временной ряд, или ряд динамики (time series), – это последовательность точек данных какого-либо события или процесса, наблюдаемого в последовательные моменты времени. Эти точки данных, часто собранные с постоянными интервалами, образуют последовательность дискретных метрик, характеризующих изменения в последовательности с течением времени. Например, график фондовой биржи изображает временной ряд, соответствующий изменению стоимости определенной фондовой единицы (акции, облигации, ценной бумаги и т. п.) по времени. Подобным образом команды оболочки bash, вводимые в командной строке, также могут образовать временной ряд. В этом случае точки данных не всегда могут быть равнозначно позиционированы во времени. Тогда последовательность является управляемой событиями (event-driven), т. е. каждое событие – это команда, выполняемая в командной оболочке. Тем не менее мы будем рассматривать и такие потоки данных, как временной ряд, поскольку каждая точка данных связана с определенным временем возникновения соответствующего события.

Изучение методов выявления аномалий тесно связано с концепцией анализа временны́х рядов, так как аномалию часто определяют как отклонение от того, что принято считать обычным, или ожидаемым, а также наблюдаемым в прошлом. Таким образом, изучение аномалий с учетом течения времени приобретает особый смысл. В следующих разделах дается определение, что такое выявление аномалий, подробно описывается процесс генерации временны́х рядов, рассматриваются конкретные методики выявления аномалий в потоке данных.

Когда следует использовать методы выявления аномалий вместо обучения с учителем

Как было отмечено в главе 1, выявление аномалий часто объединяют с распознаванием образов, например при использовании обучения с учителем, из-за этого иногда не ясно, какой подход следует применить при разработке решения конкретной задачи. Например, если проводится поиск мошеннических транзакций с помощью кредитной карты, то, возможно, имеет смысл воспользоваться моделью обучения с учителем, если имеется большой объем корректных и мошеннических транзакций, на которых можно тренировать эту модель. Обучение с учителем в особенности подходит для подобной задачи, если предполагается появление в будущем экземпляров мошеннических транзакций, похожих на примеры мошенничества, которые уже имеются в тренировочном наборе. Компании,

выпускающие кредитные карты, иногда специально ищут и сохраняют характерные шаблоны (образцы), которые, вероятнее всего, будут появляться в мошеннических транзакциях, а не в корректных: например, покупки на крупные суммы, сделанные после ряда мелких приобретений, оплата покупок из необычной локации или приобретение продукции, не соответствующей обычному профилю расходов клиента. Такие образцы (шаблоны) могут быть извлечены из большого набора положительных и отрицательных тренировочных примеров с помощью обучения с учителем.

Во многих других ситуациях может оказаться затруднительным поиск репрезентативного набора положительных примеров, достаточного для того, чтобы алгоритм «понял», что представляют собой положительные события. Причиной уязвимостей сервера иногда становятся атаки «нулевого дня», или ошибки в новых версиях программного обеспечения. По определению метод вторжения невозможно предсказать заранее, поэтому трудно сформировать профиль каждого возможного метода проникновения в систему. Поскольку такие события возникают относительно редко, это также влияет на несбалансированность классификации проблемы, следовательно, затрудняет применение обучения с учителем. Для проблем такого рода наилучшим решением является методика выявления аномалий.

Выявление вторжений с эвристиками

Системы выявления вторжений (intrusion detection systems – IDS)[1] существуют с 1986 г. и широко применяются в средах с системами обеспечения безопасности. Даже в наши дни использование пороговых значений, эвристик и простых статистических профилей остается надежным способом выявления вторжений и аномалий. Например, предположим, что в качестве верхнего предела нормального режима использования некоторой конкретной базы данных определено значение «10 запросов в час». При каждом запросе к этой базе данных вызывается функция `is_anomaly(user)` с передачей в нее идентификатора пользователя как аргумента. Если запрос к базе данных от этого пользователя выполняется в 11-й раз в течение часа, то функция помечает этот факт доступа как аномалию[2].

Несмотря на то что логика выявления аномалий на основе порогового значения проста в реализации, все же сразу возникают некоторые вопросы. Как установить правильное пороговое значение? Можно ли установить для некоторых пользователей более высокое пороговое значение по сравнению с другими пользователями? Возможны ли интервалы времени, в которые пользователям на законном основании требуется более частый доступ к базе данных? Как часто необходимо обновлять пороговое значение? Может ли атакующий извлечь данные, захватив управление многими учетными записями пользователей, т. е. для вторжения потребуется меньше сеансов доступа от каждой учетной записи?

---

[1] *Dorothy Denning*. An Intrusion-Detection Model. IEEE Transactions on Sotware Engineering SE-13:2(1987): 222–232.

[2] См. исходный код этого примера *chapter3/ids_heuristics_a.py* (**https://github.com/oreilly-ml-sec/book-resources/blob/master/chapter3/ids_heuristics_a.py**) в репозитории кода для данной книги.

Скоро мы убедимся в том, что машинное обучение может помочь избежать необходимости отвечать на все приведенные выше вопросы, позволив самим данным определить решение подобной задачи.

Возможно, в первую очередь потребуется попытка сделать метод выявления более надежным и устойчивым с помощью замены жестко заданного порогового значения «10 запросов в час» на пороговое значение, динамически генерируемое из имеющихся данных. Например, можно вычислять скользящую среднюю количества запросов от каждого пользователя в день и при каждом обновлении скользящей средней устанавливать пороговое значение в час как скорректированное кратное скользящих средних в день. (Приемлемое кратное значение может быть 5/24, т. е. почасовое пороговое значение в пять раз превышает среднечасовое.)

Возможны следующие усовершенствования этой методики:

- поскольку для аналитика данных, вероятнее всего, запросы данных потребуются гораздо чаще, чем для обычного оператора-регистратора, можно классифицировать пользователей по ролям и установить для каждой роли различные пороговые значения числа запросов;
- вместо обновления порогового значения с помощью средних значений, которые легко подделать, можно воспользоваться другими статистическими свойствами исследуемого набора данных. Например, если используется медиана, или интерквартильный размах (вероятное отклонение), то пороговые значение должны быть более устойчивым к промахам и допускать только осторожные и осмысленные попытки вмешательства в целостность системы.

Описанный выше метод использует простой статистический сбор данных, чтобы избежать необходимости ручного определения порогового значения, но сохраняет при этом характеристики эвристического метода, включая необходимость принятия решения по произвольным параметрам, таким как `avg_multiplier`. Но в решении с адаптивным пороговым значением мы уже начинаем замечать главные признаки машинного обучения детектора аномалий. Параметр `query_treshold`[1] похож на параметр модели, извлекаемый из набора данных о регулярных событиях, а цикл обновления почасового порогового значения – это непрерывный процесс тренировки, необходимый для адаптации системы к изменениям пользовательских потребностей.

Но в системах, подобных описанной выше, продолжают наблюдаться определенные недостатки. В подобной простой искусственно созданной среде сопровождение одного параметра порогового значения, обучающегося по единственному признаку, существующему в системе (число запросов от одного пользователя в час), является вполне приемлемым решением. Но в системах, которые хоть немного сложнее, количество вычисляемых пороговых значений может быстро выходить из-под контроля. Могут даже существовать обобщенные сценарии, в которых аномалии возникают не при переходе через одно пороговое значение, а при превышении определенного сочетания нескольких пороговых значений, выбира-

[1] См. исходный код этого примера *chapter3/ids_heuristics_b.py* (**https://github.com/oreilly-ml-sec/book-resources/blob/master/chapter3/ids_heuristics_b.py**) в репозитории кода для данной книги.

емых отдельно в различных ситуациях. В некоторых случаях возможна даже абсолютная неприемлемость использования детерминированного набора условий. Если пользователь А выполняет 11 запросов в час, а пользователь Б – 99 запросов в час, должны ли мы назначить более высокую оценку потенциальной опасности пользователю Б, а не пользователю А? Вероятностный подход может иметь больший смысл и позволит оценить правдоподобие того, что событие действительно является аномальным, в отличие от принятия бинарных решений.

Методы, управляемые данными

Прежде чем начать подробное изучение различных методов выявления аномалий, важно определить все цели оптимальной системы выявления аномалий:

- низкий уровень ложноположительных и ложноотрицательных результатов: термин «аномалия» подразумевает событие, которое явно выделяется на фоне прочих событий. С учетом этого факта может показаться неинтуитивным предположение, что поиск аномалий часто напоминает поиск белого кролика во время снежной бури. Из-за трудностей надежного определения «нормальности» с помощью набора дескриптивных признаков аномалии возникают в такой форме, что их легко перепутать с ложными сигналами (ложноположительные результаты) или пропустить настоящий сигнал тревоги (ложноотрицательные результаты).
  Ложноотрицательные результаты возникают, когда система не обнаружила ни одного элемента, который намеревались найти пользователи. Предположим, что вы установили в свою входную дверь новый замок с выдвижным ригелем, который устроен так, что пресекает 9 из 10 попыток взлома. Как вы оцените эффективность этого замка? Ложноположительные случаи возникают, когда система ошибочно считает обычные события аномалиями. Если вы попытаетесь отодвинуть ригель замка своим ключом, а замок не позволит это сделать, посчитав вас взломщиком, это и есть ложноположительный результат.
  Ложноположительные результаты могут выглядеть не слишком опасными, и энергично действующая система обнаружения, которая «избегает риска» и генерирует предупреждения даже при малейшем подозрении на аномалии, может решить, что такие случаи не относятся к некорректным вариантам. Но каждое предупреждение связано с издержками, поэтому каждый ложный сигнал расходует ценное время человека-аналитика, который вынужден исследовать подобные случаи. Высокий уровень ложных сигналов может быстро привести к нарушению целостности всей системы, и аналитики больше не смогут воспринимать сигнал об аномалиях как событие, требующее быстрой ответной реакции и тщательного изучения. Оптимальный детектор аномалий должен точно определять все аномалии без ложноположительных результатов;
- простота конфигурирования, тонкой настройки и сопровождения: как мы уже убедились, конфигурирование систем выявления аномалий может быть далеко не простой задачей. Неправильная конфигурация систем на основе пороговых значений становится непосредственной причиной возникновения ложноположительных и ложноотрицательных случаев. При этом даже при небольшом количестве параметров точной настройки вы теряе-

те доверие пользователей, вынужденных часто возвращаться к значениям по умолчанию (если это возможно) или выбирать случайные значения. На удобство пользования системой существенно влияет простота начальной конфигурации и долговременное сопровождение. Машинно обучаемый детектор аномалий, размещенный в вашей сети на длительное время, может начать генерировать ложные сигналы с высокой частотой, вынуждая оператора расследовать эти случаи. Оптимальный детектор аномалий должен предоставлять четкую картину того, как изменение параметров системы напрямую приводит к изменениям качества, количества и природы выходных сигналов;

- адаптируемость к изменениям трендов в данных: сезонные колебания, или просто сезонность (seasonality), – это свойство данных показывать постоянно повторяющиеся шаблоны (образцы), возникающие из-за естественных циклов активности пользователей (например, регулярная низкая активность в конце недели). Сезонные колебания необходимо учитывать во всех временны́х рядах систем распознавания образов (шаблонов), и детекторы аномалий не являются исключениями. Разные наборы данных имеют различные характеристики, но многие наборы демонстрируют один и тот же тип сезонности с меняющейся периодичностью. Например, веб-трафик, исходящий из основной временно́й зоны, будет соответствовать суточному шаблону с пиками в дневное время и минимумами ночью. На большинстве веб-сайтов наблюдается более высокий уровень трафика в будние дни по сравнению с выходными, в то время как на других сайтах можно видеть обратную картину. Некоторые тренды сезонных колебаний существуют в течение долгого времени. Веб-сайты онлайновой торговли ожидают резкое повышение трафика каждый год во время сезонов интенсивного шопинга, тогда как трафик на веб-сайте Налогового управления США (IRS) возрастает в период между январем и апрелем, после чего резко уменьшается.

  Алгоритмы выявления аномалий, которые не имеют механизма обработки сезонных колебаний, будут выдавать большое количество ложноположительных срабатываний в периоды сезонных трендов, которые необходимо отличать от ранее полученных данных. Неизбежное отклонение (drift) в данных, вызываемое лавинным распространением или постепенным ростом популярности определенных объектов, также может стать причиной сигналов об опасности, генерируемых детекторами аномалий для событий, не требующих вмешательства человека. В идеале система выявления аномалий должна быть способной идентифицировать и изучать все тренды в данных и соответствующим образом регулировать свою работу в подобных случаях при выявлении промахов;
- эффективная работа с наборами данных различной природы: несмотря на то что распределение Гаусса является основным инструментом во многих областях статистики, не все наборы данных соответствуют распределению Гаусса. В действительности лишь для нескольких задач по выявлению аномалий в сфере обеспечения безопасности подходят модели, использующие распределение Гаусса. Оценка плотности (density estimation) является основной концепцией при моделировании нормальности для выявления

аномалий, но существуют и другие ядра (kernels)[1], которые могут оказаться более подходящими для моделирования распределения конкретного набора данных. Например, для некоторых наборов данных больше подходят экспоненциальное, цилиндрическое (tophat), косинусное ядро или ядро Епанечникова (**https://scikit-learn.org/stable/modules/density.html**). Идеальная система выявления аномалий не должна делать никаких предположений о данных, т. е. должна работать одинаково хорошо с данными, обладающими разнообразными свойствами;

- эффективность использования ресурсов и возможность использования в приложениях реального времени: именно в сфере обеспечения безопасности выявление аномалий является задачей, в которой необходимо контролировать время выполнения (скорость реакции). Операторы должны быть предупреждены о потенциальных уязвимостях или аварийных ситуациях в системе в течение нескольких минут с момента возникновения подозрительных сигналов. На счету каждая секунда, когда атакующий стремится проникнуть в систему. Следовательно, такая система выявления аномалий должна работать в непрерывном потоковом режиме, собирая и обрабатывая данные и генерируя выводы с минимальными задержками. Это требование исключает некоторые слишком медленные методики и/или методики с интенсивным потреблением ресурсов;
- объяснимые сигналы тревоги: исследование и анализ сигналов тревоги, генерируемых детекторами аномалий, важно для оценки системы, а также для определения ложноположительных и ложноотрицательных срабатываний. Можно с легкостью анализировать сигналы тревоги, приходящие от детектора аномалий на основе статического порогового значения. Простое повторное возникновение события по правилам, определяемым механизмом, точно сообщает, какие условия стали причиной сигнала тревоги. Но для адаптивных систем и машинно-обучаемых детекторов аномалий эта задача становится более сложной. Когда нет явных границ решения для параметров системы, иногда становится затруднительным точно определить конкретное свойство события, которое является причиной сигнала тревоги. Невозможность объяснения затрудняет отладку и точную настройку систем и снижает уверенность в правильности решений, принятых механизмом выявления. Проблема объяснимости является предметом активных исследований в области машинного обучения, в частности для парадигмы выявления аномалий. Но когда сигналы тревоги необходимо анализировать в среде с жесткими временны́ми требованиями, наличие четких объяснений может существенно упростить процесс принятия

[1] Ядро (kernel) – это предоставляемая алгоритму машинного обучения функция, которая определяет степень схожести двух вводов (входных данных). Ядра предлагают альтернативный подход к конструированию признаков: вместо извлечения отдельных признаков из необработанных («сырых») данных функции ядра могут быть эффективно вычисляемыми, иногда в пространстве с высокой размерностью, для генерации неявных признаков из данных. При любых альтернативных подходах генерация признаков была бы связана с чрезмерными издержками. Методика эффективного преобразования данных в пространство неявных признаков с высокой размерностью известна под названием «трюк с ядром» (kernel trick). Более подробно об этом *см*. главу 2.

решений человеком или машинными компонентами, которые реагируют на сигналы об аномалиях.

Конструирование признаков для выявления аномалий

Как и в любой другой задаче машинного обучения, выбор правильных признаков для выявления аномалий имеет первостепенную важность. Многие онлайновые (потоковые) алгоритмы выявления аномалий требуют ввода в форме временно́й (динамической) последовательности потока данных. Если источник данных уже обеспечивает вывод метрик в такой форме, то, возможно, какие-либо дополнительные операции по конструированию признаков не нужны. Например, для определения того, что некоторый системный процесс чрезвычайно интенсивно использует процессор, необходима только метрика использования ЦПУ (CPU utilization), которую можно получить из многих штатных модулей мониторинга системы. Но в большинстве случаев на практике потребуется генерация специально сформированных потоков данных, к которым будут применяться алгоритмы выявления аномалий.

В этом разделе основное внимание уделено обсуждению конструирования признаков в трех областях: выявление вторжения на хост (host intrusion detection), выявление вторжения в сеть (network intrusion detection) и выявление вторжения в веб-приложение (web application intrusion detection). Между этими тремя областями имеются существенные различия, и в каждой из них требуется набор специфических условий, определяемых конкретной сферой деятельности. Мы рассмотрим примеры инструментальных средств, которые можно использовать для извлечения характерных признаков, а также обсудим достоинства и недостатки различных методик извлечения признаков.

Разумеется, область применения методов выявления аномалий не ограничивается хостами и сетями. Другие варианты практического применения, такие как выявление мошенничества и выявление аномалий в открытых вызовах прикладных программных интерфейсов (API), также основаны на правильном извлечении признаков с целью получения надежного источника данных, пригодного для применения соответствующих алгоритмов. После подробного рассмотрения принципов извлечения полезных признаков и временны́х рядов данных из рабочей среды хостов и сетевых доменов вы сможете применить на практике эти принципы в конкретных прикладных областях.

Выявление вторжения на хост

При разработке агента, выявляющего вторжения на хосты (серверы, настольные системы, ноутбуки, встроенные системы), вероятнее всего, потребуется генерация собственных специализированных метрик. Возможно, даже возникнет необходимость в установлении взаимосвязей между сигналами, собираемыми из различных источников. Значимость различных метрик может изменяться в широких пределах в зависимости от модели потенциальной опасности, но основные статистические данные уровня системы и сети являются хорошим отправным пунктом. Набор этих системных метрик можно формировать различными способами, существует большое разнообразие инструментов и рабочих сред для решения подобной задачи. Мы рассмотрим osquery (**https://osquery.io/**) – широко

известную инструментальную рабочую среду для операционной системы, позволяющую собирать и просматривать метрики низкого уровня ОС и предоставлять к ним доступ с помощью запросов через интерфейс на основе языка SQL. Создание запланированных запросов с помощью osquery может помочь в установке основного рабочего режима и нормального поведения хоста и/или приложения, следовательно, позволит детектору вторжений определять подозрительные события, возникающие неожиданно.

Вредоносное программное обеспечение является самым распространенным вектором угроз для хостов в большинстве операционных сред. Разумеется, выявление и анализ вредоносного программного обеспечения требует отдельного подробного рассмотрения, которое будет представлено в главе 4. Но сейчас наш анализ основан на том предположении, что большая часть вредоносных программ воздействует на операции системного уровня, следовательно, обнаружить вредоносную программу можно с помощью сбора сигналов об операциях системного уровня и поиска признаков угрозы (indicators of compromise – IoC) в этих данных. Ниже приведен список некоторых общих сигналов, которые можно собирать для решения подобной задачи:

- работающие (активные) процессы;
- активные/новые учетные записи пользователей;
- загружаемые модули ядра;
- операции поиска DNS;
- сетевые соединения;
- изменения в механизме системного планировщика;
- фоновые/постоянные (резидентные) процессы и демоны;
- операции запуска (активизации), операции launchd;
- регистрационные базы данных ОС, файлы .plist;
- каталоги для временных файлов;
- расширения браузера.

Этот список далеко не полон, поскольку различные типы вредоносного ПО обычно придерживаются разнообразных линий поведения, но сбор обширного диапазона сигналов гарантирует, что вы получите более полное представление о работе тех частей системы, для которых потенциальная опасность воздействия вредоносного ПО наиболее высока.

osquery

В инструментальной рабочей среде osquery можно планировать периодическое выполнение запросов с помощью демона osqueryd и заполнение полученными данными таблиц, которые в дальнейшем можно использовать для подробного изучения. В исследовательских целях также можно выполнять оперативные запросы, воспользовавшись интерфейсом командной строки osqueryi. Пример запроса, выдающего список всех пользователей системы, приведен ниже:

```
SELECT * FROM users;
```

Если необходимо определить пять процессов с неумеренным потреблением оперативной памяти, то запрос будет выглядеть следующим образом:

```
SELECT pid, name, resident_size FROM processes
ORDER BY resident_size DESC LIMIT 5
```

Рабочую среду osquery можно использовать для наблюдения за сохранением надежности и согласованности системы, но одним из главных способов применения этой среды является определение поведения (рабочего режима) компонентов в системе, для которой существует потенциальная угроза вторжения. Вредоносный бинарный файл обычно пытается свести к минимуму или вовсе уничтожить следы своей деятельности в системе, в том числе и в файловой системе, например после начала выполнения такой файл удаляет сам себя. Общий запрос для поиска аномальных выполняющихся бинарных файлов позволяет проверить текущие активные процессы, у которых удалены соответствующие выполняемые файлы:

```
SELECT * FROM processes WHERE on_disk = 0;
```

Предположим, что этот запрос сгенерировал данные, выглядящие следующим образом:

```
2017-06-04T18:24:17+00:00        []
2017-06-04T18:54:17+00:00        []
2017-06-04T19:24:17+00:00        ["/tmp/YBBHNCA8J0"]
2017-06-04T19:54:17+00:00        []
```

Самым простым способом преобразования этих данных в числовой временной ряд является использование длины элементов списка как определяющего значения. После этого становится ясно, что третий элемент списка в этом примере будет зарегистрирован как аномалия.

Помимо наблюдения за состоянием системы, демон osqueryd также может отслеживать события уровня ОС, такие как изменения в файловой системе и факты доступа к ней, операции монтирования дисков, изменения состояния процессов, изменения настроек сети и многое другое. Это позволяет организовать тщательное наблюдение в ОС, управляемой событиями, для сохранения целостности файловой системы и осуществления контроля процессов и сокетов.

В osquery включены удобные инстументы – пакеты запросов (query packs) – комплекты запросов и метрик, сгруппированные по предметным областям, и практические примеры использования, которые пользователи могут загружать и применять для демона osqueryd. Например, пакет incident-response позволяет наблюдать метрики, относящиеся к защитному экрану приложений, crontab, IP-форвардингу, iptables, launchd, прослушиваемым портам, операциям монтирования дисков, операциям открытия файлов и сокетов, истории командной оболочки, записям о запуске программ и т. д. Пакет osx-attacks позволяет искать особые сигналы, характерные для набора известных вредоносных программ в macOS, с проверкой заданных списков plists, имен процессов или приложений, о которых известно, что они являются компонентами, устанавливаемыми вредоносным ПО.

Настройка osquery может выполняться в созданном файле конфигурации, в котором определяется, какие запросы и пакеты должна использовать система[1]. Например, можно запланировать выполнение запроса каждые полчаса (т. е. через каждые 1800 секунд). Запрос выявляет удаленные бинарные файлы работающих программ. Для этого в файле конфигурации необходимо разместить следующий код запроса:

[1] Пример файла конфигурации можно найти в репозитории osquery на GitHub (**https://github.com/osquery**).

```
{
 ...
  // Планирование запросов для периодического выполнения:
  "deleted_running_binary": {
    "query": "SELECT * FROM processes WHERE on_disk = 0;",
    "interval": 1800
  }
  ...
}
```

Следует отметить, что osquery может фиксировать в журнале результаты запросов как мгновенные снимки или протоколы различий. Журналирование протоколов различий может быть полезным для сокращения объема получаемой информации, но такая форма журнальных записей более сложна для синтаксического анализа (парсинга). После того как демон заносит в журнал эти данные, извлечение временны́х рядов метрик превращается в простую задачу анализа файлов журналов или выполнения дополнительных SQL-запросов к сгенерированным таблицам.

Ограничения osquery с точки зрения обеспечения безопасности

Важно понимать, что osquery не предназначался для работы в ненадежных средах. В osquery нет встроенных функций для маскировки и защиты операций и журнальных записей, поэтому существует потенциальная опасность, что вредоносная программа вмешается в процесс сбора метрик или в операции записи в журналы/базу данных, чтобы уничтожить свои следы. Развернуть osquery на одном хосте очень просто, но в большинстве крупных организаций более вероятно наличие множества серверов с разнообразными функциями, развернутых в различных операционных средах. В osquery нет встроенных функциональных возможностей для оркестровки или централизованного развертывания и управления рабочей средой, поэтому потребуются дополнительные усилия и затраты по разработке с целью интегрирования osquery в рабочие среды автоматизации и оркестровки конкретной организации (например, можно воспользоваться инструментальными средствами Chef (**https://www.chef.io/chef/**), Puppet (**https://puppet.com/**), Ansible (**https://www.ansible.com/**), SaltStack (**https://saltstack.com/**)). Кроме того, существует большое количество инструментальных средств сторонних производителей, предназначенных для упрощения работы с osquery, например Kolide (**https://kolide.co/**) для распределенного управления и выполнения команд osquery, и doorman (**https://github.com/mwielgoszewski/doorman**) – менеджер, управляющий распределенными экземплярами osquery.

Другие возможности и инструменты

Существует множество решений как коммерческих, так и с открытым исходным кодом, представляющих собой альтернативу osquery. Эти решения могут помочь достигнуть того же конечного результата: непрерывное и подробное наблюдение за хостами. Анализ огромного объема информации, который многие Unix-подобные системы предоставляют штатными средствами (например, содержимое подкаталога /proc), – это простейшее решение, которое может оказаться вполне достаточным для конкретного практического случая. Система аудита Linux Auditing System (auditd и т. п.) является более зрелым и совершенным инструментальным средством, чем osquery. Это инструмент, которому опытные эксперты и профессиональные специалисты доверяют в течение нескольких десятилетий.

Выявление вторжения в сеть

Почти все формы вторжения на хост провоцируют обмен информацией с внешней средой. Большинство уязвимостей используется с целью похищения важ-

ных данных с целевого хоста, поэтому имеет смысл сосредоточиться на выявлении вторжений в сетевую среду. При атаке ботнетов удаленные серверы оперативного управления и контроля обмениваются информацией со взломанными компьютерами-«зомби» для передачи инструкций по выполнению различных операций. При целевых атаках (APT) взломщики могут получить удаленный доступ к компьютерам через уязвимый или неправильно сконфигурированный сервис, позволяющий воспользоваться командной оболочкой и/или правами доступа суперпользователя (root). Рекламное ПО требует обмена информацией с внешними серверами для принудительной загрузки рекламного контента на целевой компьютер. Шпионское ПО чаще всего передает результаты скрытого мониторинга по сети на внешний принимающий сервер.

Программная экосистема выявления вторжения в сеть предлагает множество утилит и комплектов приложений, которые могут помочь в сборе сигналов из сетевого трафика любых типов: от простых утилит мониторинга протоколов, таких как tcpdump (**https://www.tcpdump.org/manpages/tcpdump.1.html**), до более сложных инструментов сниффинга, таких как Zeek (**https://www.zeek.org/**) (более известно под старым именем Bro). Работа инструментальных средств выявления вторжения в сеть основана на концепции наблюдения за трафиком, передаваемым между хостами. Как и при выявлении вторжений на хосты, атаки можно идентифицировать по совпадению трафика с известной сигнатурой (образцом) вредоносного трафика или при помощи методики выявления аномалий, сравнивающей трафик с предварительно установленными базовыми или опорными линиями. В этом разделе основное внимание уделено выявлению аномалий, а не поиску совпадений сигнатур, который будет рассматриваться в главе 4 при изучении глубокого анализа вредоносных программ.

Snort (**https://www.snort.org/**) – это широко известная система выявления вторжения (IDS) с открытым исходным кодом, которая выполняет мониторинг (сниффинг) пакетов и сетевого трафика для выявления аномалий в реальном времени. Это действительно хороший вариант выбора для организации мониторинга с целью выявления вторжений. В Snort успешно сочетаются удобство использования и функциональность. Кроме того, программа разрабатывается и поддерживается чрезвычайно активным сообществом пользователей и программистов ПО с открытым исходным кодом, которые создали многочисленные дополнения и графические интерфейсы для нее. Snort имеет относительно простую архитектуру, позволяющую пользователям выполнять анализ трафика в реальном времени в IP-сетях, формировать правила, срабатывающие при обнаружении определенных условий, и сравнивать трафик с предварительно установленной базовой (опорной) линией профиля нормальной работы сетевой среды.

При получении признаков для выявления вторжения в сеть особого внимания заслуживает различие между извлечением метаданных сетевого трафика и наблюдением за содержимым сетевого трафика. Извлечение метаданных сетевого трафика используется при инспекции пакетов с сохранением состояния (stateful packet inspection – SPI), работающей на сетевом и транспортном уровнях (в модели OSI это уровни 3 и 4 (**https://www.webopedia.com/quick_ref/OSI_Layers.asp**)) и исследующей заголовок и трейлер каждого сетевого пакета без обращения к содержимому. При этой методике сохраняется состояние предыдущих принятых пакетов, следовательно, есть возможность устанавливать связь между новыми принятыми

и ранее исследованными пакетами. SPI-системы способны определять, является ли пакет частью процедуры обмена рукопожатиями для установления нового соединения, частью уже существующего сетевого соединения или непредусмотренным мошенническим пакетом. Такие системы полезны при строгом контроле доступа – обычная функция сетевых экранов (network firewalls), – поскольку они дают четкую картину IP-адресов и портов, участвующих в обмене данными. Также SPI-системы могут оказаться полезными при выявлении несколько более сложных атак на 3/4 уровнях[1], таких как IP-спуфинг, TCP/IP-атаки (например, повреждение ARP-кеша или SYN-флудинг) и атаки типа DoS (отказ сервиса). Тем не менее очевидны недостатки этой методики, потому что анализ ограничен только заголовками и трейлерами пакетов. Например, SPI-система не способна определить признаки уязвимостей (брешей) или вторжение на уровне приложений, поскольку для этого требуется более глубокий уровень исследований.

Полная инспекция пакетов

Полная инспекция пакетов (deep packet inspection – DPI) – это процесс исследования данных, содержащихся в сетевых пакетах в дополнение к инспекции заголовков и трейлеров. Такой подход позволяет накапливать сигналы и статистические данные о сетевой корреспонденции на уровне приложений. Поэтому методика полной инспекции пакетов способна собирать сигналы, которые могут помочь в выявлении спама, вредоносного ПО, вторжений и менее заметных аномалий. Полная инспекция пакетов в потоковом режиме реального времени представляет собой весьма трудную ИТ-задачу, так как требует немалых вычислительных мощностей для дешифровки, дизассемблирования и анализа пакетов, передаваемых в различных сетевых средах.

Bro (новое название Zeek, *см*. выше) является одной из самых первых систем, предлагающих реализацию рабочей среды пассивного мониторинга сети для выявления вторжений. Bro состоит из двух компонентов: эффективный механизм обработки событий, извлекающий сигналы из текущего сетевого трафика в реальном времени, и механизм стратегии (policy engine), принимающий и обрабатывающий события и скрипты стратегий и выполняющий соответствующее действие в ответ на различные полученные сигналы.

Один из возможных вариантов практического применения Bro – выявление подозрительной активности в веб-приложениях с помощью инспекции строк в теле HTTP-запросов POST. Например, можно выявить случаи внедрения SQL-кода и межсайтового скриптинга (XSS) с помощью создания профиля содержимого тела запроса POST для точки входа в конкретное веб-приложение. Сигнал тревоги может быть сгенерирован при обнаружении определенных аномальных символов (символ ' (апостроф) в случае внедрения SQL-кода и символы скриптовых тегов `<or>` в случае межсайтового скриптинга) в процессе сравнения с базовой линией. Такие сигналы могут оказаться весьма полезными для выявления атак злоумышленников на конкретное веб-приложение[2].

---

[1] *Frédéric Cuppens et al.* Handling Stateful Firewall Anomalies. Proceedings of the IFIP International Information Security Conference (2012): 174–186.

[2] *Ganesh Kumar Varadarajan.* Web Application Attack Analysis Using Bro IDS. SANS Institute (2012).

Набор признаков, генерируемых с помощью методики полной инспекции пакетов, для выявления аномалий в чрезвычайно высокой степени зависит от природы приложений, работающих в вашей сетевой среде, а также от векторов потенциальных угроз, характерных для вашей инфраструктуры. Если сеть не содержит открытых для внешнего мира веб-серверов, то использование полной инспекции пакетов для выявления атак межсайтового скриптинга не имеет смысла. Если в сети имеются только системы POS-терминалов, соединенные с базами данных PostgreSQL, хранящими данные клиентов, возможно, следует сосредоточиться на выявлении непредусмотренных сетевых соединений, которые могут быть признаками того, что атакующий проник в вашу сеть.

Такие техники взлома и нелегального вторжения, как pivoting и island hopping, представляют собой многоуровневые стратегии атаки, используемые злоумышленниками для обхода защитных сетевых экранов. Правильно сконфигурированная сеть не позволит получить доступ извне к важным секретным базам данных. Но если в сети имеется уязвимый компонент, открытый для всеобщего доступа, с внутренним каналом доступа к важной базе данных, то атакующие могут взломать такой компонент и последовательно пройти («прыгать» – hop) к серверам базы данных через промежуточные узлы (используя их косвенно как опоры для «прыжков» – island hopping). В зависимости от наличия открытых портов и разрешенных протоколов между взломанным и целевым хостами атакующие могут воспользоваться различными методами для создания точки опоры (pivoting). Например, атакующий может установить прокси-сервер на взломанном хосте, тем самым создав скрытый туннель между целевым хостом и внешней сетевой средой.

Если методика полной инспекции пакетов используется в среде с применением протоколов Transport Layer Security/Secure Sockets Layer (TLS/SSL), где проверяемые пакеты зашифрованы, то приложение, реализующее полную инспекцию пакетов, обязательно должно ограничить действие SSL. Полная инспекция пакетов непременно требует, чтобы система выявления аномалий работала по схеме «инспектор в середине маршрута» (man-in-the-middle), т. е. чтобы данные, передаваемые через пункт инспекции, уже не были защищены на протяжении всего пути (end-to-end). Такая архитектура может создавать серьезные угрозы безопасности и/или производительности в вашей рабочей среде, особенно в случаях, когда ограничение действия SSL и повторное зашифрование пакетов реализовано неправильно. Необходимо с особой тщательностью исследовать и проверить методики конструирования признаков, которые перехватывают TLS/SSL-трафик, прежде чем развертывать их в реальной эксплуатационной среде.

Признаки для выявления вторжений в сеть

Рабочая группа Knowledge Discovery and Data Mining Special Interest Group (SIGKDD) организации Association of Computing Machinery (ACM) ежегодно поддерживает конкурс KDD Cup, предлагая разнообразные сложные задачи для участников. В 1999 г. предлагалась тема «выявление вторжений в компьютерные сети» (computer network intrusion detection – **https://www.kdd.org/kdd-cup/view/kdd-cup-1999**), по которой была поставлена задача «обучения модели прогнозирования, способной различать законные (допустимые) и незаконные (недопустимые) соединения в компьютерной сети». Искусственно созданный набор данных для этой задачи очень стар, и в нем имеются существенные недостатки и упущения, но список признаков, выведенных с помощью этого набора данных, представляет собой не-

плохой исходный пример конструирования признаков для выявления вторжений в сеть в любой конкретной среде. Штодмайер (Staudemeyer) и Омлин (Omlin) использовали этот набор данных для определения наиболее важных признаков[1]. Их работа может оказаться полезной при определении типов признаков, которые необходимо генерировать для выявления аномалий и вторжений в сеть. Группировка транзакций по IP-адресам, географическому положению, подсетям (например, /16, /14), префиксам BFP, по информации о номерах автономных систем (ASN) и т. п. часто может являться вполне подходящим способом для выделения самых значимых характеристик сложных сетевых сред и генерации простых числовых метрик для выявления аномалий[2].

Выявление вторжений в веб-приложение

Ранее мы видели, что можно выявить атаки на веб-приложение, такие как межсайтовый скриптинг и внедрение SQL-кода с использованием инструментов полной инспекции сетевых пакетов, например Bro (Zeek). Инспекция журналов HTTP-сервера может предоставить аналогичный уровень информации и представляет собой более очевидный способ получения признаков, выводимых из взаимодействий пользователей с веб-приложением. Стандартные веб-серверы, такие как Apache, IIS и Nginx, генерируют журнальные записи в общем формате NCSA Common Log Format (**https://en.wikipedia.org/wiki/Common_Log_Format**), которые также называют журналами доступа (access logs). В формате NCSA объединенные журналы и журналы ошибок также регистрируют информацию о User-agent клиента, о соответствующем URL и обо всех ошибках сервера, вызванных запросами. В этих журналах каждая строка представляет отдельный HTTP-запрос, обращенный к серверу, и состоит из токенов в строго определенном формате. Ниже приведен пример записи в формате объединенного журнала, включающей User-agent автора запроса и соответствующий URL:

```
123.123.123.123 - jsmith [17/Dec/2016:18:55:05 +0800] "GET /index.html HTTP/1.0"
200 2046 "http://referer.com/" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10.17.3)
AppleWebKit/536.27.14 (KHTML, like Gecko) Chrome/55.0.2734.24 Safari/536.27.14"
```

В отличие от методики полной инспекции пакетов, стандартные журналы доступа через веб не регистрируют данные тела запроса POST при настройках по умолчанию. Таким образом, векторы атаки, включенные во входные данные пользователя, не могут быть обнаружены с помощью инспектирования стандартных журналов доступа.

Самые распространенные веб-серверы предоставляют дополнительные модули и подключаемые компоненты, которые позволяют регистрировать в журналах содержимое HTTP-запросов. Для сервера Apache предлагается модуль `mod_dumpio` (**https://httpd.apache.org/docs/2.4/mod/mod_dumpio.html**), который фиксирует в журнале все принимаемые входные данные и выходные данные, передаваемые сервером. В файл конфигурации Nginx можно добавить директивы `proxy_pass` или `fastcgi_pass`, чтобы заставить серверы Nginx явно при-

[1] *Ralf Staudemeyer and Christian Omlin*. Extracting Salient Features for Network Intrusion Detection Using Machine Learning Methods. South African Computer Journal 52 (2014): 82–96.

[2] *Alex Pinto*. Applying Machine Learning to Network Security Monitoring. Black Hat webcast presented May 2014.

сваивать переменной `$request_body` значение в виде содержимого тела запроса POST. Компания Microsoft снабжает свои серверы IIS расширением Advanced Logging (**https://www.microsoft.com/ru-ru/download/details.aspx?id=7211**), которое можно сконфигурировать так, чтобы в журналах регистрировались данные запроса POST.

Даже при сравнительно ограниченном уровне видимости содержимого запросов в стандартных файлах журналов HTTP-сервера в них все же имеются некоторые полезные признаки, которые можно извлечь:

- статистические данные о доступе на уровне IP: высокая частота, периодичность или большой объем данных, передаваемых с одного IP-адреса или из одной подсети, вызывают подозрение;
- искажение строки URL: пути со ссылками на себя (/./) или на родительские каталоги (/../) часто используются в атаках с использованием пересечения путей файловой системы;
- декодированные элементы URL и HTML, экранированные символы, завершение строки нулевым байтом – эти приемы нередко используются простыми механизмами формирования подписи/правил, для того чтобы избежать обнаружения;
- необычные ссылочные шаблоны: доступ к странице с необычно выглядящей ссылкой на URL часто является сигналом о несанкционированном доступе к конечному пункту HTTP;
- последовательно выполняемые попытки доступа к конечному пункту: беспорядочные попытки доступа к конечным пунктам HTTP, которые не соответствуют логическому потоку выполнения на веб-сайте, являются признаками фаззинга (искажения входных данных) или злонамеренных вторжений.

  Например, если обычная попытка доступа пользователя к веб-сайту представляет собой запрос POST в /login, за которым следуют три последовательных запроса GET в /a, /b и /c, но конкретный IP-адрес многократно выполняет запросы GET в /b и /c без соответствующих запросов в /login и/или /a, это может быть признаком автоматизированной работы бота или шпионским зондированием, выполняемым вручную;
- шаблоны User-agent: можно выполнять частотный анализ строк User-agent для предупреждений о появлении ранее не наблюдаемых строк User-agent или о чрезвычайно старых клиентах (например, User-agent `"Mosaic/0.9"` 1993 года), которые, вероятнее всего, являются поддельными.

Журналы веб-серверов предоставляют достаточный объем информации для выявления различных типов атак на веб-приложения[1], в том числе (но не только) атак из первой десятки OWASP Top Ten (**https://www.owasp.org/index.php/Category:OWASP_Top_Ten_Project**) – межсайтовый скриптинг (XSS), внедрение SQL-кода, CSRF, Insecure Direct Object References и т. д.

Краткие итоги

Генерация надежного и полного набора признаков чрезвычайно важна для процесса выявления аномалий. Целью конструирования признаков является пред-

[1] *Roger Meyer*. Detecting Attacks on Web Applications from Log Files. SANS Institute (2008).

ставление сложно организованной информации в простой компактной форме с исключением ненужной информации, но при этом нельзя терять какие-либо важные характеристики данных. Затем эти сгенерированные признаки передаются в алгоритмы, которые принимают и обрабатывают данные, после чего используют их для тренировки моделей машинного обучения. В следующем разделе рассматриваются возможности преобразования наборов признаков в конкретные прогнозы (предположения), которые управляют системами выявления аномалий.

Выявление аномалий с помощью данных и алгоритмов

После завершения конструирования набора признаков из исходного потока событий для генерации временны́х рядов наступает время использования алгоритмов для генерации предположений (прогнозов) из этих данных. Выявление аномалий имеет долгую историю в области академических научных исследований, но, как и во всех прочих прикладных областях анализа данных, здесь не существует единственного универсального алгоритма, применимого для всех типов временны́х рядов. Поэтому следует ожидать, что процесс поиска наилучшего алгоритма для каждого конкретного приложения будет представлять собой процесс исследований и экспериментов.

Перед выбором алгоритма важно учесть природу (сущность) и качество источника данных. Данные, значительно искаженные аномалиями, вряд ли повлияют на методологию выявления. Как утверждалось выше в текущей главе, если данные не содержат аномалий (или содержат помеченные аномалии, которые можно с легкостью удалить), то это задача выявления новизны (novelty detection). В противном случае задача называется выявлением промахов (outlier detection). При выявлении промахов выбранный алгоритм должен быть нечувствительным к малым отклонениям, которые снижают качество тренируемой модели. Часто определение подходящей методики требует нетривиального решения. Чистка набора данных с целью удаления аномалий – очень трудная, а иногда совершенно невозможная задача. Если неизвестно, содержат ли ваши данные какие-либо аномалии, возможно, лучше всего начать с предположения, что аномалии имеются, и постепенно улучшать решение.

В этом разделе мы попытаемся объединить многочисленные разнообразные методы выявления аномалий[1], взятые из специальной литературы и промышленной практики, в общую схему категоризации на основе основных принципов каждого алгоритма. При этом каждый алгоритм должен принадлежать не более чем одной категории. Классификация выполняется по следующим категориям:

- прогнозирование (машинное обучение с учителем);
- статистические метрики;
- машинное обучение без учителя;
- проверки точности (качества) подгонки;
- методы на основе плотности.

[1] В этом разделе термины «алгоритм», «метод» и «методика» используются как взаимозаменяемые, но все они обозначают один из специализированных способов реализации выявления аномалий, например унарная классификация методом опорных векторов или эллиптическая огибающая.

Каждая категория характеризуется собственным подходом к задаче поиска аномалий. Будут рассмотрены достоинства и недостатки каждой методики и степень пригодности различных наборов данных для одной из методик по сравнению с другими. Например, прогнозирование подходит только для одномерных временны́х рядов данных, тогда как методы на основе плотности более пригодны для многомерных наборов данных.

Приводимый здесь обзор не является полным, кроме того, мы не даем подробные описания теоретической основы и реализации каждого алгоритма. Это скорее общий обзор некоторых различных вариантов, предлагаемых для практической реализации конкретных систем выявления аномалий, которыми, как мы надеемся, вы сможете воспользоваться для разработки оптимального решения.

Прогнозирование (машинное обучение с учителем)

Прогнозирование – это интуитивно понятный способ реализации выявления аномалий: выполняется обучение на ранее существующих данных, затем формируется прогноз будущих событий на основе новых данных. Можно рассматривать любые существенные различия между прогнозами и наблюдениями как аномалии. Если взять пример с прогнозом погоды, то при отсутствии дождя в течение нескольких недель и явных признаков будущего дождя можно дать прогноз с малой вероятностью дождя в следующие дни. Если в следующие дни все-таки пошел дождь, то это следует считать отклонением от сделанного прогноза.

Этот класс алгоритмов выявления аномалий использует ранее накопленные данные для прогнозирования по текущим данным и дает числовую оценку того, на какие величины текущие наблюдаемые данные отличаются от спрогнозированных. В соответствии с этим определением прогнозирование относится к классу машинного обучения с учителем, поскольку тренирует регрессионную модель данных, сравнивая значения различного времени. Поскольку эти алгоритмы также оперируют только понятиями прошлого и настоящего, они пригодны только для одномерных наборов данных в форме временны́х рядов. Предсказания, сделанные по модели прогнозирования, будут соответствовать ожидаемому значению, которое конкретный временной ряд будет иметь на следующем шаге во времени. Таким образом, применение прогнозирования к наборам данных, отличающимся от временны́х рядов, не имеет смысла.

Данные временных рядов естественным образом могут быть представлены в виде графика с одной линией. Людям удобно изучать такие графики с единственной линией, определяя тренды и выявляя аномалии, но машинам (компьютерам) труднее работать с таким представлением данных. Основной причиной этого затруднения является шум, присутствующий в данных временно́го ряда из-за неточности измерений, дискретизации выборки или других внешних факторов, присущих самой природе данных. Шум приводит к формированию нестабильных, беспорядочно изменяющихся рядов, которые могут скрывать возмущения или пиковые максимумы, которые необходимо идентифицировать. С учетом сезонности и циклических шаблонов (образцов), которые иногда могут быть весьма сложными, попытка использования простейших «линейных» методов для выявления аномалий, вероятнее всего, не принесет удовлетворительных результатов.

При прогнозировании важно определить следующие характеристики временны́х рядов:

- тренды – длительная ориентация изменений в данных, не нарушаемая относительно мелкомасштабными отклонениями и возмущениями. Иногда тренды бывают нелинейными, но обычно хорошо аппроксимируются полиномиальными кривыми низкого порядка;
- сезоны – периодические повторения шаблонов в данных, обычно соответствующие факторам, тесно связанным с природой данных. Например, ночные и дневные шаблоны (образцы), различия в летний и зимний периоды, фазы луны;
- циклы – общие изменения в данных, обладающие сходством по шаблонам, но различающиеся по периодичности, например долговременные циклы на рынке ценных бумаг.

На рис. 3.1 показана сезонность с суточным шаблоном и слабо возрастающим трендом, изображенным в виде линии регрессии, соответствующей этим данным.

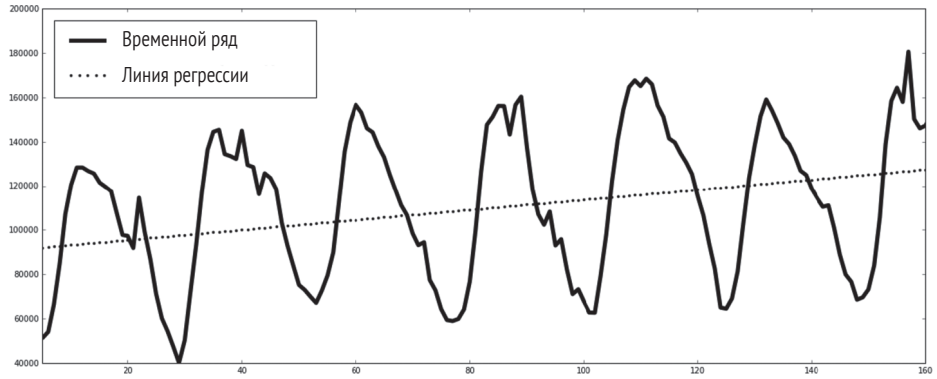


Рис. 3.1 ❖ Ежесуточная сезонность и возрастающий тренд

ARIMA

Использование семейства функций ARIMA (autoregressive integrated moving average – интегрированная модель авторегрессии – скользящего среднего) – мощный и гибкий способ прогнозирования по временны́м рядам. Модели авторегрессии – это класс статистических моделей, результаты работы которых линейно зависимы от их собственных предшествующих значений в сочетании со стохастическим фактором[1]. Возможно, вы имеете представление об экспоненциальном сглаживании (exponential smoothing), которое часто можно приравнять или аппроксимировать особыми частными случаями ARIMA (например, экспоненцальное сглаживание Холта–Уинтерса (Holt–Winters)). Эти операции сглаживают зубчатые линейные графики, используя различные варианты взвешенных скользящих средних для нормализации данных. Сезонные варианты таких операций могут рассматривать периодически повторяющиеся шаблоны, способствуя формированию более точных прогнозов. Например, сезонная методика ARIMA (SARIMA)

[1] Строго говоря, автокорреляция (autocorrelation) – это корреляция вектора временного ряда с тем же вектором, смещенным на некоторый отрицательный небольшой интервал времени (дельта времени).

определяет и сезонный, и внесезонный компоненты модели ARIMA, позволяя получить периодические характеристики[1].

При выборе подходящей модели прогнозирования необходимо всегда выполнять визуализацию данных, чтобы определить тренды, сезонность и циклы. Если сезонность является преобладающей характеристикой рядов данных, то следует рассмотреть модели с учетом сезонности, такие как SARIMA и сезонные методы Холта–Уинтерса. Методы прогнозирования изучают характеристики временны́х рядов, исследуя предшествующие точки и формируя прогнозы на будущее. При исследовании данных полезной метрикой для обучения является автокорреляция (autocorrelation), представляющая собой корреляцию между исследуемым рядом данных и тем же рядом данных в некоторый предшествующий момент времени. Правильный прогноз для ряда данных можно считать будущими точками данных с высокой автокорреляцией с предшествующими точками данных.

ARIMA использует модель с распределенным лагом (distributed lag model), в которой регрессии применяются для прогнозирования будущих значений на основе лаговых переменных (значений) (авторегрессивный процесс). Параметры авторегрессии и скользящего среднего используются для точной настройки модели вместе с методом факторизации многочленов с разностью степеней – процесс, применяемый для того, чтобы сделать последовательности стационарными (stationary), т. е. обладающими постоянными статистическими свойствами во времени, например меаной (средним) и дисперсией. Это свойство ARIMA требует для всех входных последовательностей (рядов).

В следующем примере сделана попытка выявить аномалии по ежеминутно определяемой метрике использования процессора хоста[2]. На рис. 3.2 по оси y показан процент использования процессора, по оси x – время.

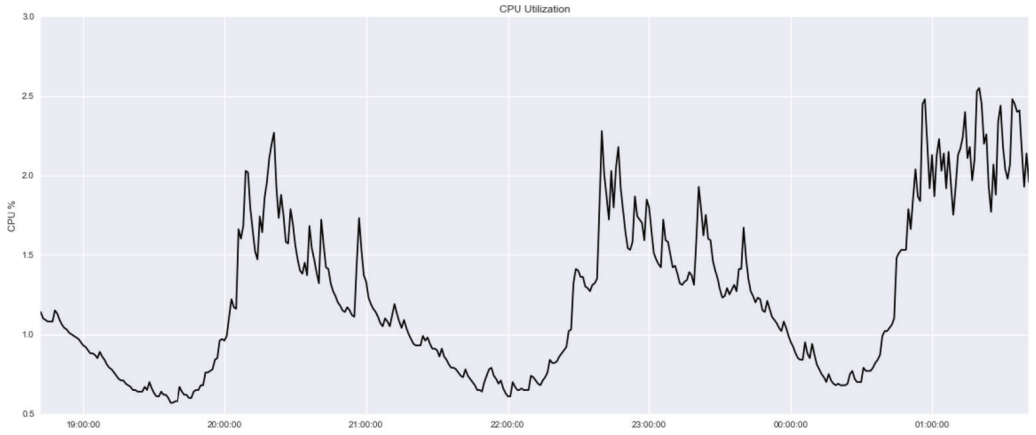


Рис. 3.2 ❖ График использования процессора с течением времени

[1] Роберт Нау (Robert Nau) из Duke University предоставляет мощный, подробно проработанный ресурс для прогнозирования, ARIMA и многого другого (**https://people.duke.edu/~rnau/411home.htm**).

[2] См. chapter3/datasets/cpu-utilization (**https://github.com/oreilly-mlsec/book-resources/tree/master/chapter3/datasets/cpu-utilization**) в репозитории кода для этой книги.

В этой последовательности можно наблюдать явный периодически повторяющийся шаблон с пиками использования процессора приблизительно через каждые 2,5 часа. Используя удобную библиотеку обработки временны́х рядов для языка Python PyFlux (**https://github.com/RJT1990/pyflux**; **https://pyflux.readthedocs.io/en/latest/**), мы применяем алгоритм прогнозирования ARIMA с порядком авторегрессии (AR) 11, порядком скользящего среднего (MA) 11 и порядком разности 0 (поскольку ряд уже выглядит стационарным). Существует несколько особых приемов определения порядков AR, MA (**https://people.duke.edu/~rnau/411arim3.htm**) и разности (**https://people.duke.edu/~rnau/411arim2.htm**), которыми мы здесь не пользуемся. Для чрезвычайно простых случаев порядки AR и MA требуют корректировки всех остаточных автокорреляций, сохраняющихся в различных рядах (т. е. между собственно рядом и тем же рядом, чуть сдвинутым во времени). Порядок разности – это параметр, используемый для перевода рядов в стационарное состояние. Таким образом, уже стационарные ряды должны иметь порядок разности 0, ряды с постоянным трендом среднего (с постоянно повышающимся или понижающимся трендом) должны иметь порядок разности 1, а ряды с изменяющимся во времени трендом (с трендом, изменяющимся по скорости и направлению в конкретном ряду) должны иметь порядок разности 2. Попробуем построить соответствующий выборке график, чтобы понять принцип работы алгоритма[1]:

```
import pandas as pd
import pyflux as pf
from datetime import datetime

# Считывание файлов тренировочного и тестового наборов данных
data_train_a = pd.read_csv('cpu-train-a.csv',
    parse_dates=[0], infer_datetime_format=True)
data_test_a = pd.read_csv('cpu-test-a.csv',
    parse_dates=[0], infer_datetime_format=True)

# Определение модели
model_a = pf.ARIMA(data=data_train_a,
                   ar=11, ma=11, integ=0, target='cpu')

# Оценка скрытых переменных для этой модели с использованием
# алгоритма Метрополиса–Хастингса как метода выведения значений
x = model_a.fit("M-H")

# Построение графического отображения модели ARIMA, соответствующего исследуемым данным
model_a.plot_fit()
```

На рис. 3.3 показан итоговый график ряда выборки.

На рис. 3.3 можно видеть, что результат прогноза достаточно точно совпадает с исследуемыми данными. Далее можно выполнить тестирование выборки по последним 60 точкам тренировочных данных. Тестирование выборки представляет собой этап проверки (валидации), на котором обрабатывается последняя (по времени) часть ряда как ранее неизвестная и выполняется прогнозирование для

[1] Полный код примера можно найти в Python Jupyter notebook chapter3/arima-forecasting.ipynb (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter3/arima-forecasting.ipynb**) в репозитории кода для этой книги.

этого интервала времени. Этот процесс позволяет оценить эффективность модели без выполнения тестирования на будущих/тестовых данных:

```
> model_a.plot_predict_is(h=60)
```

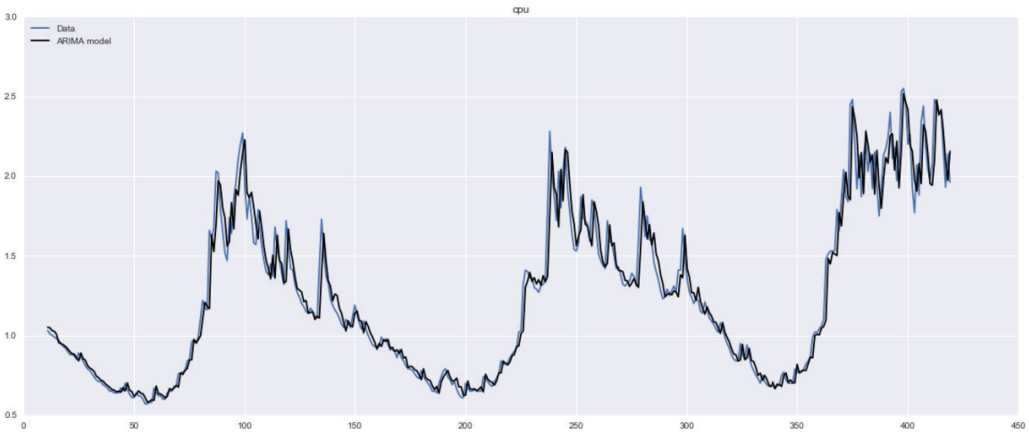


Рис. 3.3 ❖ График использования процессора во времени, совмещенный с прогнозом по модели ARIMA

Тест прогнозирования в рамках выборки (показанный на рис. 3.4) выглядит достаточно точным, так как не содержит существенных отклонений от исходного ряда по фазе и амплитуде.

Рис. 3.4 ❖ Прогнозирование ARIMA в пределах выборки (на тренировочном наборе данных)

Теперь переходим к настоящему прогнозированию, отображая на графике 100 самых последних наблюдаемых точек данных, расположенных после 60 спрогнозированных моделью значений, позиционированных с достоверными (доверительными) интервалами:

```
> model_a.plot_predict(h=60, past_values=100)
```

Полосы, закрашенные более темным цветом, представляют более высокую достоверность, как показано на рис. 3.5.

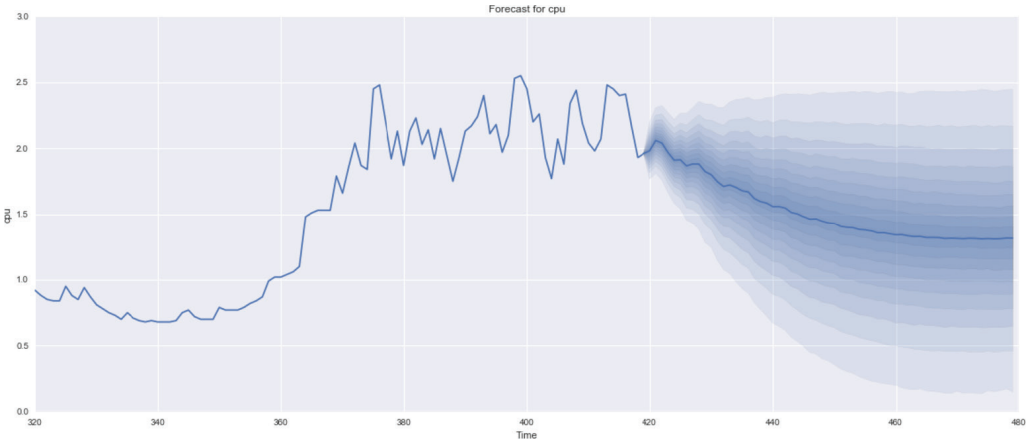


Рис. 3.5 ❖ Прогнозирование ARIMA за пределами выборки (на тестовом наборе данных)

Сравнивая прогноз, показанный на рис. 3.5, с действительно наблюдаемыми точками, изображенными на рис. 3.6, можно убедиться, что прогноз точный.

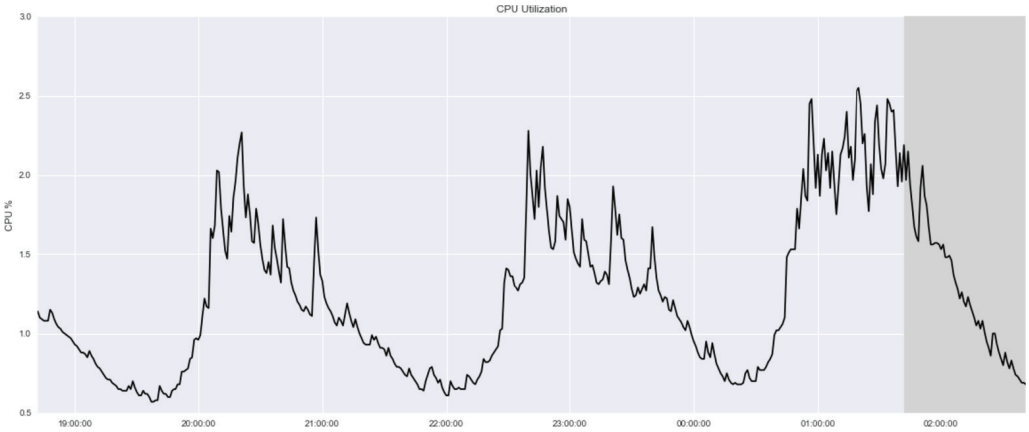


Рис. 3.6 ❖ Действительно наблюдаемые точки данных

Для выполнения выявления аномалий с использованием прогнозирования реально наблюдаемые точки данных сравниваются с динамическим прогнозом, выполняемым периодически. Например, любая произвольная, но разумно организованная система может выполнять прогноз на 60-минутный интервал каждые 30 минут, тренируя новую модель ARIMA с помощью данных, накопленных в предыдущие 24 часа. Сравнение прогноза и наблюдений может происходить гораздо более часто (например, каждые три минуты). Этот метод пошагового (инкрементного) обучения можно применять почти ко всем алгоритмам, которые бу-

дут рассматриваться в дальнейшем, поскольку это позволяет аппроксимировать динамическое (потоковое, непрерывное) поведение по алгоритмам, изначально предназначенным для пакетной (статической) обработки.

Выполним некоторые операции прогнозирования на другом сегменте набора данных об использовании процессора, собранных в другой интервал времени:

```
data_train_b = pd.read_csv('cpu-train-b.csv',
    parse_dates=[0], infer_datetime_format=True)
data_test_b = pd.read_csv('cpu-test-b.csv',
    parse_dates=[0], infer_datetime_format=True)
```

При прогнозировании с использованием той же модели ARIMAX[1], тренированной на наборе данных `data_train_b`, итоговый прогноз показан на рис. 3.7.

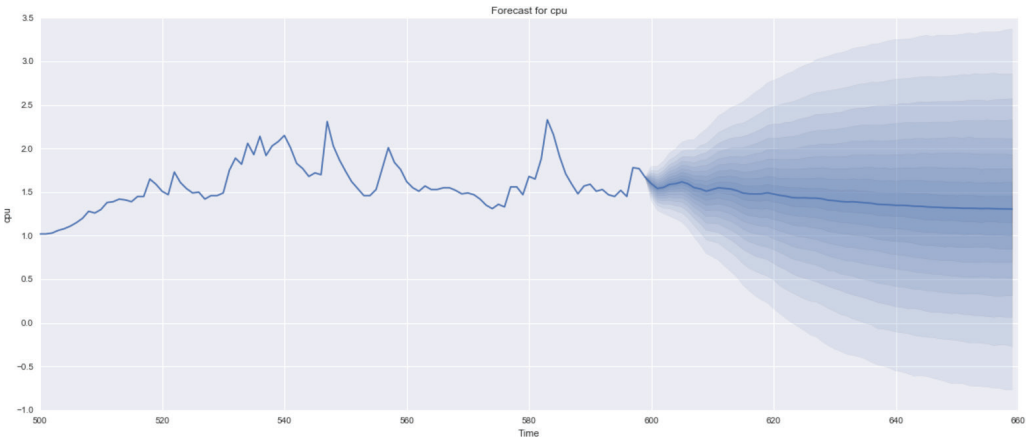


Рис. 3.7 ❖ Прогноз с помощью модели ARIMAX за пределами выборки (тестовый набор, data_train_b)

Но в этом случае наблюдаемые значения существенно отличаются от прогнозируемых, как показано на рис. 3.8.

Здесь наблюдается явная аномалия, возникающая в коротком интервале времени после фазы тренировки. Поскольку резкое уменьшение наблюдаемых значений расположено в зоне низкой достоверности, будет сгенерировано оповещение об аномалии. Обязательно должны быть установлены специальные пороговые условия, определяющие, на какую величину отличаются прогнозируемые и наблюдаемые ряды, для генерации сигнала об аномалии. Эти условия специфичны для конкретного приложения, но они должны быть достаточно простыми в реализации.

---

[1] ARIMAX – это слегка модифицированная модель ARIMA, в которую добавлены компоненты из стандартной эконометрики, известные как объясняющие (независимые) переменные для моделей прогнозирования.

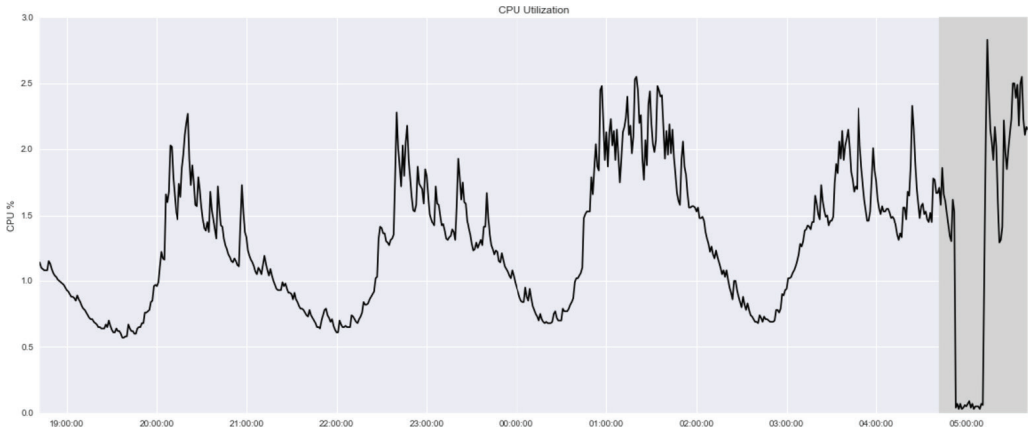


Рис. 3.8 ❖ Действительно наблюдаемые точки данных (data_train_b)

Искусственные нейронные сети

Другим способом прогнозирования по временны́м рядам данных является использование искусственных нейронных сетей. В частности, сети типа долгая краткосрочная память (long short-term memory – LSTM)[1] хорошо подходят для этого приложения. LSTM-сети являются вариантом рекуррентных нейронных сетей (RNN) с особой архитектурой, специально предназначенной для обучения на трендах и шаблонах во временны́х рядах как входных данных для задач классификации и прогнозирования. Здесь мы не будем рассматривать теорию и подробности реализации нейронных сетей, а просто применим их как черные ящики, способные обучаться на информации из временны́х рядов, содержащих шаблоны, которые встречаются с неизвестной или нерегулярной периодичностью. Для этого будет использоваться API Keras LSTM (**https://keras.io/layers/recurrent/#lstm**), поддерживаемый библиотекой с открытым исходным кодом TensorFlow (**https://www.tensorflow.org/**), для прогнозирования на том же наборе данных об использовании процессора, с которым мы работали ранее.

Методика обучения для этой LSTM-сети вполне понятна. Сначала извлекаются все непрерывные подмножества данных длины n из тренировочного входного набора. Самая последняя точка в каждом подмножестве определяется как метка данной выборки. Другими словами, из входных данных генерируются n-граммы (n-grams). Например, если принять n = 3 для необработанных входных данных:

```
raw: [0.51, 0.29, 0.14, 1.00, 0.00, 0.13, 0.56],
```

то можно получить следующие n-граммы (триграммы):

---

[1] *Sepp Hochreiter and Jürgen Schmidhuber*. Long Short-Term Memory. Neural Computation 9 (1997): 1735–1780.
Alex Graves. Generating Sequences with Recurrent Neural Networks. University of Toronto (2014) (**https://arxiv.org/pdf/1308.0850v5.pdf**).

```
n-grams: [[0.51, 0.29, 0.14],
          [0.29, 0.14, 1.00],
          [0.14, 1.00, 0.00],
          [1.00, 0.00, 0.13],
          [0.00, 0.13, 0.56]]
```

В результате тренировочный набор будет выглядеть так, как показано в табл. 3.1.

Таблица 3.1

Выборка	Метка
(0.51, 0.29)	0.14
(0.29, 0.14)	1.00
(0.14, 1.00)	0.00
(1.00, 0.00)	0.13
(0.00, 0.13)	0.56

Выбранная модель обучается для прогнозирования третьего значения в последовательности, следующего за двумя уже известными, ранее наблюдаемыми значениями. LSTM-сети немного сложнее в практическом применении из-за работы с запоминаемыми шаблонами и сохраняемой информацией из предыдущих последовательностей, но, как уже было отмечено выше, мы не будем углубляться в подробности. Ниже приведено определение четырехслойной[1] LSTM-сети[2]:

```
from keras.models import Sequential
from keras.layers.recurrent import LSTM
from keras.layers.core import Dense, Activation, Dropout

# Каждая точка тренировочных данных должна иметь длину 100-1,
# потому что самое последнее значение в каждой последовательности является меткой
sequence_length = 100

model = Sequential()

# Первый слой LSTM, определяющий длину входной последовательности
model.add(LSTM(input_shape=(sequence_length-1, 1),
               units=32,
               return_sequences=True))
model.add(Dropout(0.2))

# Второй слой LSTM со 128 элементами
model.add(LSTM(units=128,
               return_sequences=True))
model.add(Dropout(0.2))
```

---

[1] Нейронные сети состоят из слоев (layers) отдельных элементов. Данные передаются во входной слой, а прогнозы выполняются в выходном слое. Между этими слоями может быть произвольное число скрытых слоев (hidden layers). При подсчете числа слоев в нейронной сети учитывается общепринятое соглашение: не считать входной слой. Например, в шестислойной нейронной сети имеется один входной слой, пять скрытых слоев и один выходной слой.

[2] Полный код примера можно найти в Python Jupyter notebook chapter3/lstm-anomaly-detection.ipynb (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter3/lstm-anomaly-detection.ipynb**) в репозитории кода для этой книги.

```
# Третий слой LSTM со 100 элементами
model.add(LSTM(units=100,
               return_sequences=False))
model.add(Dropout(0.2))

# Плотно связанный выходной слой с линейной функцией активации
model.add(Dense(units=1))
model.add(Activation('linear'))
model.compile(loss='mean_squared_error', optimizer='rmsprop')
```

Точная архитектура сети (число слоев, размер каждого слоя, тип слоя и т. д.) выбирается произвольно с приближением на основе других известных LSTM-сетей, хорошо работающих при решении подобных задач. Следует отметить добавление члена `Dropout(0.2)` после каждого скрытого слоя – это исключение (dropout)[1], метод регуляризации, который широко используется для предотвращения переподгонки нейронных сетей. В конце определения модели выполняется вызов метода model.compile() (**https://keras.io/models/model/#compile**), который конфигурирует процесс обучения. Параметр оптимизации rmsprop optimizer (**https://keras.io/optimizers/#rmsprop**) выбран, потому что в документации утверждается, что обычно это наилучший выбор для RNN. В процессе подгонки модели будет использоваться алгоритм оптимизации rmsprop для минимизации функции потерь, которая определена как mean_squared_error (**https://keras.io/losses/#mean_squared_error**). Существует множество других точно настраиваемых параметров и разнообразных архитектур, улучшающих эффективность модели, но, как всегда, мы предпочитаем простоту точности.

Подготовка входных данных:

```
...
# Генерация n-грамм из исходных последовательностей тренировочного набора данных
n_grams = []
for ix in range(len(training_data)-sequence_length):
n_grams.append(training_data[ix:ix+sequence_length])

# Нормализация и перемешивание значений
n_grams_arr = normalize(np.array(n_grams))
np.random.shuffle(n_grams_arr)

# Отделение каждой выборки от ее метки
x = n_grams_arr[:, :-1]
labels = n_grams_arr[:, -1]
...
```

Затем можно прогнать данные через модель и выполнить прогнозирование:

```
...
model.fit(x,
   labels,
   batch_size=50,
   nb_epochs=3,
   validation_split=0.05)

y_pred = model.predict(x_test)
...
```

[1] *Nitish Srivastava et al.* Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15 (2014): 1929–1958.

На рис. 3.9 показаны результаты вместе со среднеквадратическим отклонением (RMS).

Здесь можно видеть, что прогноз располагается достаточно близко к неаномальному наблюдаемому ряду (оба нормализованы), и это позволяет считать, что созданная LSTM-сеть действительно работает успешно. При возникновении наблюдаемых аномалий мы видим значительное отклонение прогнозируемого ряда от наблюдаемого, доказательством этого служит график RMS. Как и в случае применения ARIMA, такие размеры отклонений между прогнозом и наблюдаемыми данными могут использоваться для генерации сигнала о выявлении аномалий. Установление пороговых значений для расхождения между наблюдаемым и прогнозируемым рядами является хорошим способом исключения незначительных различий в данных и простой методикой числовой оценки непредвиденных отклонений.

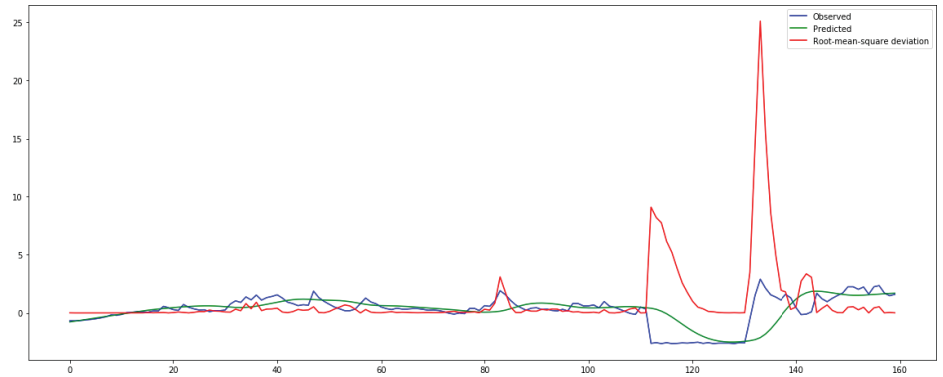


Рис. 3.9 ❖ Графики наблюдаемого ряда данных, прогноза и среднеквадратичного отклонения LSTM-сети для выявления аномалий во временно́м ряду использования процессора

Краткие итоги

Прогнозирование – это интуитивно понятный метод реализации выявления аномалий. Особенно если временной ряд содержит предсказуемые шаблоны сезонности и явно наблюдаемый тренд, такие модели, как ARIMA, способны собирать и накапливать данные и делать надежные прогнозы. Для более сложных временны́х рядов данных могут лучше подойти LSTM-сети. Существуют и другие методики прогнозирования, которые используют те же принципы и достигают той же цели. Восстановление временного ряда данных из тренировочной модели машинного обучения (например, из модели кластеризации) может быть использовано для генерации прогноза, но справедливость и обоснованность такого подхода стала причиной обсуждений в академических кругах[1].

Следует отметить, что методики прогнозирования обычно не работают удовлетворительно для выявления промахов, т. е. если тренировочные данные для выбранной модели содержат аномалии, которые невозможно отфильтровать без

[1] *Eamonn Keogh and Jessica Lin.* Clustering of Time-Series Subsequences Is Meaningless: Implications for Previous and Future Research. Knowledge and Information Systems 8 (2005): 154–177.

затруднений, то эта модель будет соответствовать и правильным данным, и промахам одновременно, что чрезвычайно затрудняет выявление будущих промахов. Такая методика хорошо подходит для выявления новизны, т. е. для тех случаев, когда аномалии содержатся только в тестовых данных, но не в тренировочном наборе. Если временной ряд содержит чрезвычайно много ошибочных данных и для него невозможно определить какой-либо очевидный тренд или если амплитуда отклонений изменяется в широких пределах, то, вероятнее всего, прогнозирование не будет успешно выполнено. Лучше всего прогнозирование работает на одномерных рядах реально оцениваемых метрик, поэтому если набор данных содержит многомерные векторы признаков или категориальных переменных, то необходимо выбрать другую методику выявления аномалий.

Статистические метрики

Можно использовать статистические тесты для определения сходства одной новой точки данных с ранее наблюдаемыми данными. Пример в начале текущей главы, в котором был создан детектор аномалий на основе пороговых значений, адаптированный к изменениям данных с помощью сохранения и накопления скользящего среднего для временно́го ряда, попадает в эту категорию. Можно воспользоваться скользящими средними временны́х рядов данных как адаптивной метрикой, которая показывает, насколько хорошо точки данных соответствуют долговременному тренду. В частности, скользящее среднее (также известное под названием фильтр нижних частот – ФНЧ; low-pass filter – из терминологии обработки сигналов) является контрольной точкой для статистических сравнений. Значительные отклонения от этого среднего значения будут считаться аномалиями. В этом разделе кратко рассматриваются некоторые заслуживающие внимания метрики, но каждой метрике не будет уделяться слишком много внимания, поскольку их практическое применение вполне понятно и не вызывает затруднений.

Медиана абсолютного отклонения

Стандартное отклонение последовательности данных часто используется при установлении адаптивного порогового значения для выявления аномалий. Например, разумное определение некоторой аномалии может быть любой точкой, которая расположена на расстоянии более двух стандартных отклонений от среднего значения. Таким образом, для набора данных со стандартным нормальным распределением (**https://en.wikipedia.org/wiki/Normal_distribution**) с математическим ожиданием 0 и стандартным отклонением 1 любая точка данных, лежащая между значениями –2 и 2, считается нормальной, тогда как точка данных со значением 2,5 должна считаться аномальной. Этот алгоритм работает, если исследуемые данные абсолютно точные, но если данные содержат промахи, то вычисленное среднее значение и стандартные отклонения будут искажены.

Медиана абсолютного отклонения (median absolute deviation – MAD) – это широко используемая альтернатива стандартному отклонению для поиска промахов в одномерных данных. Медиана абсолютного отклонения определяется как медиана абсолютных отклонений от срединного значения ряда[1]:

[1] Код этого примера можно найти в файле chapter3/mad.py (**https://github.com/oreilly-ml-sec/book-resources/blob/master/chapter3/mad.py**) в репозитории кода для данной книги.

```
import numpy as np

# Входной ряд данных
x = [1, 2, 3, 4, 5, 6]

# Вычисление медианы абсолютного отклонения
mad = np.median(np.abs(x - np.median(x)))

# MAD ряда x равна 1.5
```

Поскольку медиана (срединное значение) гораздо менее чувствительна к воздействию промахов, чем среднее значение, медиана абсолютного отклонения является надежной мерой, подходящей для использования в тех случаях, когда тренировочные данные содержат промахи.

Критерий Граббса для определения промахов

Критерий Граббса для определения промахов (**https://en.wikipedia.org/wiki/Grubbs'_test_for_outliers**) – это алгоритм, который находит один промах в наборе данных с нормальным распределением, рассматривая текущее минимальное или максимальное значение в исследуемом ряде. Алгоритм применяется итеративно с удалением предыдущего выявленного промаха перед каждой итерацией. Хотя здесь мы не углубляемся в подробности, отметим, что наиболее часто используемым способом применения критерия Граббса для выявления аномалий является вычисление статистического критерия Граббса и критического значения Граббса и пометка точки как промаха, если тестовый статистический критерий больше, чем критическое значение. Такая методика подходит только для рядов с нормальным распределением и может оказаться неэффективной, так как определяет только одну аномалию на каждой итерации.

Краткие итоги

Сравнение статистических метрик – весьма простой способ реализации выявления аномалий, хотя многие не считают его методикой машинного обучения. Тем не менее этот способ действительно обеспечивает проверку многих типов признаков для оптимального детектора аномалий, который мы обсуждали выше: сигналы об аномалиях являются воспроизводимыми, и их легко объяснить, алгоритмы адаптируются к изменениям трендов в данных. Возможно, эта методика весьма эффективна благодаря своей простоте, а ее настройка и сопровождение не вызывают никаких затруднений. Эти свойства позволяют методике сравнения статистических метрик становиться оптимальным выбором в некоторых случаях, в которых статистические характеристики можно точно вычислить или в которых приемлем более низкий уровень точности. Из-за своей простоты статистические метрики имеют ограниченные возможности обучения и часто работают хуже, чем более эффективные алгоритмы машинного обучения.

Точность аппроксимации (качество подгонки)

При создании системы выявления аномалий важно учесть, содержат ли данные, используемые для тренировки начальной модели, аномалии. Как уже было отмечено ранее, на этот вопрос трудно ответить, но часто есть возможность сделать основанное на имеющейся информации предположение, обеспечивающее правильное понимание природы источника данных и модели потенциальной угрозы.

В идеальном мире ожидаемые распределения в наборе данных могут быть точно смоделированы с помощью известных распределений. Например, распределение вызовов API, направленных на сервер приложений в день (измеряемое во времени), может быть достаточно близким к нормальному распределению, а количество атак на веб-сайт в часы после начала поддержки может быть точно описано экспоненциально убывающей последовательностью. Но поскольку мы живем не в идеальном мире, редко удается найти реальные наборы данных, в точности соответствующие простым распределениям. Даже если набор данных можно привести в соответствие некоторому гипотетическому аналитическому распределению, точно определяющему сущность этого распределения, это может оказаться чрезвычайно трудной задачей. Тем не менее такой подход вполне применим в некоторых случаях, особенно если проводится обработка большого набора данных, для которого хорошо известно ожидаемое распределение[1]. В таких случаях сравнение расхождения между ожидаемым и наблюдаемым распределениями может стать методикой выявления аномалий.

Критерии точности аппроксимации (качества подгонки или адекватности модели), такие как критерий хи-квадрат (chi-squared test), критерий (согласия) Колмогорова–Смирнова (Kolmogorov–Smirnov test) и критерий Крамера–фон Мизеса (Cramér–von Mises criterion)[2], могут использоваться для числового выражения степени схожести двух непрерывных распределений. Эти критерии в основном пригодны только для одномерных наборов данных, что существенно ограничивает их применимость. Мы не будем подробно рассматривать широко распространенные критерии точности аппроксимации как раз из-за их ограниченной применимости для реальных задач выявления аномалий. Вместо этого мы уделим основное внимание более универсальным методикам, таким как метод приближения с помощью эллиптических кривых, реализованный в библиотеке scikit-learn.

Метод приближения с помощью эллиптических огибающих кривых (приблизительная оценка ковариации)

Для наборов данных с нормальным распределением метод приближения с помощью эллиптических огибающих кривых (elliptic envelope fitting) может стать простым и элегантным способом практической реализации выявления аномалий. Поскольку по определению аномалии являются точками, которые не соответствуют ожидаемому распределению, подобные алгоритмы с легкостью исключают подобные промахи из тренировочных данных. Таким образом, эта методика в самой минимальной степени подвержена воздействию аномалий, присутствующих в исследуемом наборе данных.

При практическом применении этой методики требуется выполнить достаточно строгое предположение относительно исследуемых данных, а именно: что данные, расположенные внутри области нормальности, получены из известного аналитического распределения. Рассмотрим пример гипотетического набора данных, содержащего два промасштабированных и нормализованных соответствую-

---

[1] Закон больших чисел представляет собой теорему, утверждающую, что повторение эксперимента бесконечно большое количество раз дает средний результат, который близок к ожидаемому значению.

[2] В русскоязычной литературе его чаще называют критерием Смирнова–Крамера–фон Мизеса. – *Прим. перев.*

щим образом признака (например, пиковое значение использования процессора и время запуска процессов, инициализированных пользователем на хосте за 24 часа). Следует отметить, что такие временны́е ряды данных, соответствующие простым и общеизвестным аналитическим распределениям, на практике встречаются крайне редко. Более вероятно, что эта методика лучше подойдет для задач выявления аномалий, в которых временно́е измерение исключено. Набор данных для этого примера будет синтезирован по выборке с распределением Гаусса, поэтому имеет коэффициент содержания промахов в общем массиве 0,01:

```
import numpy as np

num_dimensions = 2
num_samples = 1000
outlier_ratio = 0.01
num_inliers = int(num_samples * (1-outlier_ratio))
num_outliers = num_samples - num_inliers

# Генерация данных, лежащих внутри области нормального распределения
x = np.random.randn(num_inliers, num_dimensions)

# Добавление промахов, выбранных из случайного равномерного распределения
x_rand = np.random.uniform(low=-10, high=10, size=(num_outliers, num_dimensions))
x = np.r_[x, x_rand]

# Генерация меток: 1 для нормальных данных, –1 для промахов
labels = np.ones(num_samples, dtype=int)
labels[-num_outliers:] = -1
```

Если изобразить этот набор данных на точечной диаграмме (рис. 3.10), то можно видеть, что промахи явно отделены от центральной моды кластера:

```
import matplotlib.pyplot as plt

plt.plot(x[:num_inliers,0], x[:num_inliers,1], 'wo', label='inliers')
plt.plot(x[-num_outliers:,0], x[-num_outliers:,1], 'ko', label='outliers')
plt.xlim(-11,11)
plt.ylim(-11,11)
plt.legend(numpoints=1)
plt.show()
```

Метод приближения с помощью эллиптических огибающих кривых кажется подходящим вариантом для выявления аномалий с учетом того, что данные выглядят нормально распределенными (как показано на рис. 3.10). Воспользуемся удобным классом `sklearn.covariance.EllipticEnvelope` (**https://scikit-learn.org/stable/modules/generated/sklearn.covariance.EllipticEnvelope.html**) для выполнения следующей операции анализа данных[1]:

```
from sklearn.covariance import EllipticEnvelope

classifier = EllipticEnvelope(contamination=outlier_ratio)
```

[1] Полный код этого примера можно найти в Python Jupyter notebook *chapter3/elliptic-envelope-fitting.ipynb* (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter3/elliptic-envelope-fitting.ipynb**) в репозитории кода для этой книги.

```
classifier.fit(x)
y_pred = classifier.predict(x)
num_errors = sum(y_pred != labels)
print('Number of errors: {}'.format(num_errors))
```

```
> Number of errors: 0
```

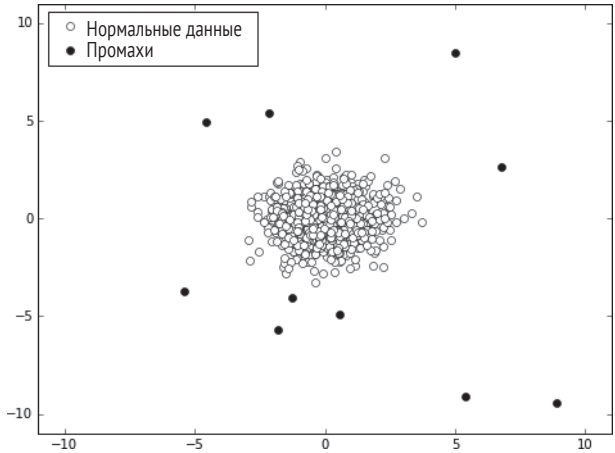


Рис. 3.10 ❖ Точечная диаграмма искусственно синтезированного набора данных с метками данных в области нормальности и промахов

Метод превосходно работает на предложенном здесь наборе данных, но это неудивительно, если учесть регулярность распределения. В нашем примере известно точное значение коэффициента `outlier_ratio`, равное 0,01, потому что этот набор данных создан искусственно. Это важный параметр, поскольку он сообщает классификатору ожидаемое пропорциональное отношение промахов и нормальных данных. В реальных практических задачах, в которых коэффициент промахов неизвестен, необходимо выбрать наилучшее предполагаемое начальное значение на основании конкретных знаний о решаемой задаче. В дальнейшем можно постепенно более точно настроить параметр `outlier_ratio`, увеличивая его, если некоторые промахи, которые алгоритм обязан находить, не определяются, или уменьшая этот параметр, если возникают ложноположительные срабатывания.

Рассмотрим более подробно сформированную этим классификатором границу решения, которая показана на рис. 3.11.

Центральная мода закрашена серым цветом и обозначена эллиптической границей решения. Любые точки, лежащие за пределами эллипса, представляющего границу решения, считаются промахами.

Необходимо помнить, что эффективность метода приближения с помощью эллиптических огибающих кривых не одинакова для различных распределений данных. Рассмотрим набор данных, не соответствующий нормальному распределению Гаусса (рис. 3.12).

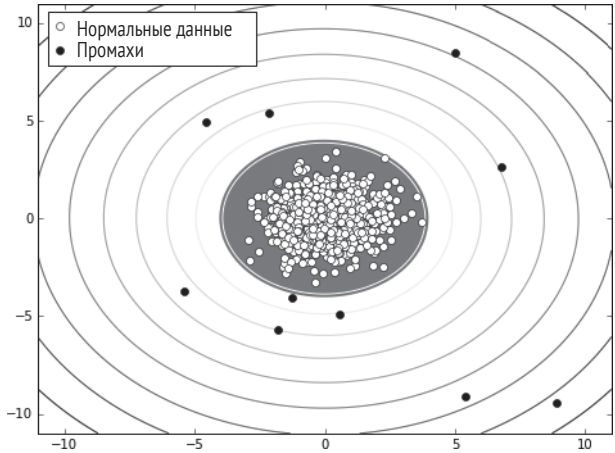


Рис. 3.11 ❖ Граница решения для метода приближения с помощью эллиптических огибающих кривых, применяемого к данным, искусственно созданным из распределения Гаусса

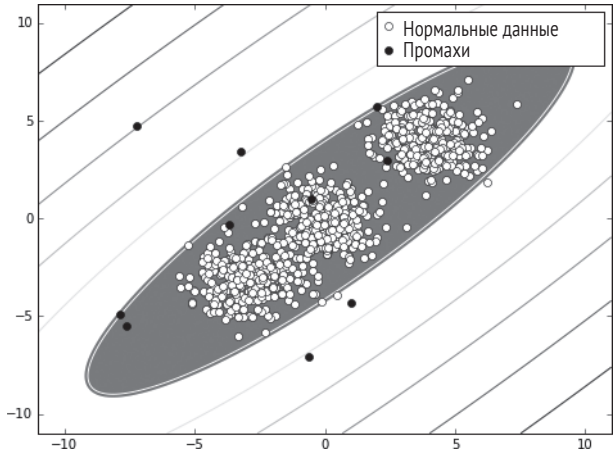


Рис. 3.12 ❖ Граница решения методом приближения с помощью эллиптических огибающих кривых для искусственно созданных данных, не соответствующих нормальному распределению Гаусса

На рис. 3.12 видны восемь неправильно классифицированных точек: четыре промаха определены как нормальные данные, а четыре нормальные точки выпали за границу решения и помечены как промахи.

Способ применения этого метода в системе динамического (потокового) выявления аномалий вполне понятен. Периодически адаптируя эллиптическую огибающую кривую к новым данным, можно получить постоянно обновляющуюся границу решения, с помощью которой классифицируются новые входящие точки данных. Для исключения эффекта девиации (возникновения погрешности – drift) и постоянного расширения границы решения со временем хорошим вариантом является изъятие из обращения некоторых точек данных по прошествии опре-

деленного интервала времени. Но чтобы при этом остались учтенными эффекты сезонности и цикличности, такое скользящее окно (sliding window) новейших точек данных должно быть достаточно широким для включения в него информации о ежедневных и еженедельных шаблонах.

Функция `EllipticEnvelope()` из библиотеки sklearn размещена в модуле `sklearn.covariance` (**https://scikit-learn.org/stable/modules/generated/sklearn.covariance.html**). Ковариация (covariance) признаков в наборе данных означает общую изменяемость (вариативность) признаков. Другими словами, это мера величины и направления эффекта, который изменяется при переходе от одного признака к другому. Ковариация – это характеристика набора данных, которую можно использовать для описания распределений, следовательно, и для выявления промахов, не соответствующих описываемым распределениям. Механизмы оценки ковариации (covariance estimators) могут быть использованы для эмпирической оценки ковариации набора данных, если взять некоторые тренировочные данные. Это в точности соответствует работе методики аппроксимации на основе ковариации для выявления аномалии.

Устойчивые (робастные) оценки ковариации[1], такие как механизм оценки MCD (Minimum Covariance Determinant; минимальный определитель матрицы ковариации), сводят к минимуму влияние промахов в тренировочных данных на модель аппроксимации. Можно измерить качество модели аппроксимации с помощью расстояния между промахами и распределением модели, используя функцию расстояния, например расстояние Махаланобиса (Mahalanobis distance). По сравнению с неустойчивыми оценками, такими как метод максимального правдоподобия (Maximum Likelihood Estimator – MLE), механизм MCD способен различать промахи и нормальные данные, генерируя более точное соответствие. В результате разделяются нормальные данные, находящиеся на небольших расстояниях от центральной моды модели аппроксимации, и промахи, более удаленные от центральной моды.

Метод приближения с помощью эллиптических огибающих кривых использует механизмы устойчивых оценок ковариации для получения оценок ковариации, моделирующих распределение регуляризированных тренировочных данных, после чего точки, не соответствующие вычисленным оценкам, классифицируются как аномалии. Мы убедились, что метод приближения с помощью эллиптических огибающих кривых достаточно хорошо работает для двумерного «загрязненного» набора данных с известным распределением Гаусса, но не так эффективен для набора данных, не соответствующего распределению Гаусса. Эту методику можно также применять к наборам данных с более высокой размерностью, но выгода от этого будет различной – эллиптические огибающие работают лучше с наборами данных с низкой размерностью. При использовании для временно́го ряда данных эта методика может оказаться полезной в некоторых случаях для удаления временно́й характеристики из набора признаков и полного соответствия моде-

[1] В статистике «устойчивый», или «робастный», – это характеристика, которая используется для описания устойчивости к промахам. В более общем смысле термин «устойчивые (робастные) статистические характеристики» обозначает статистические характеристики, которые в весьма малой степени подвержены воздействию постоянных отклонений от предполагаемых свойств модели.

ли подмножеству других признаков. Но в этом случае следует отметить отсутствие возможности захвата аномалий, статистически регулярных по отношению к агрегированному распределению, хотя в действительности аномалии зависят от времени их появления. Например, если некоторые аномальные точки данных в середине ночи имеют признаки, значения которых не отличаются от обычных точек данных в середине дня, но являются явно аномальными для измерений в ночное время, то, возможно, промахи не будут выявлены, если вы отказались от временно́го измерения.

Алгоритмы машинного обучения без учителя

Мы переходим к классу решений задачи выявления аномалий, который возник на основе модификаций обычных моделей машинного обучения с учителем. Классификаторы машинного обучения с учителем обычно применяются для решения задач, в которых существуют два и более классов. Но при использовании для выявления аномалий модификации этих алгоритмов придают им характеристики методов обучения без учителя. В этом разделе рассматривается несколько таких алгоритмов.

Метод опорных векторов с одним классом

Можно воспользоваться методом опорных векторов с одним классом для выявления аномалий, адаптируя метод опорных векторов для работы с данными, принадлежащими к единственному классу. Такие данные (предполагается, что они не содержат аномалий) используются для тренировки модели, формируя границу решения, которой можно пользоваться для классификации признаков новых входящих точек данных. Не существует устойчивого и надежного механизма, встроенного в стандартные реализации метода опорных векторов с одним классом, поэтому тренировочная модель менее устойчива к промахам в исследуемом наборе данных. Сам по себе этот метод больше подходит для выявления новизны, чем для выявления промахов, поэтому тренировочные данные в идеальном случае должны быть полностью «очищены» и не должны содержать аномалий.

Метод опорных векторов с одним классом вполне применим при работе с негауссовыми или мультимодальными распределениями (т. е. содержащими более одного «центра» нормальных данных), а также с наборами данных с высокой размерностью. Мы применим классификатор по методу опорных векторов с одним классом ко второму набору данных, который использовался в предыдущем разделе. Отметим, что этот набор данных не является идеальным для данного метода, потому что промахи составляют один процент данных, но все же попробуем узнать, в какой степени итоговая модель подвержена влиянию имеющихся «примесей»[1]:

```
from sklearn import svm

classifier = svm.OneClassSVM(nu=0.99 * outlier_ratio + 0.01,
                             kernel="rbf",
                             gamma=0.1)
```

[1] Полный код этого примера можно найти в Python Jupyter notebook chapter3/one-class-svm.ipynb (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter3/one-class-svm.ipynb**) в репозитории кода для этой книги.

```
classifier.fit(x)
y_pred = classifier.predict(x)
num_errors = sum(y_pred != labels)
print('Number of errors: {}'.format(num_errors))
```

Рассмотрим более подробно отдельные параметры, определяемые при создании объекта `svm.OneClassSVM` (**https://scikit-learn.org/stable/modules/generated/sklearn.svm.OneClassSVM.html**). Отметим, что эти параметры зависимы от конкретных наборов данных и вариантов использования. В любом случае вы должны всегда хорошо понимать значения и смысл всех настраиваемых параметров, предлагаемых классификатором, прежде чем использовать их. При работе с малой пропорциональной частью промахов в данных параметр `nu` устанавливается приблизительно равным коэффициенту промахов. В соответствии с документацией библиотеки sklearn этот параметр управляет верхней границей группы ошибок в тренировочном наборе и нижней границей группы, относящейся к опорным векторам. Другими словами, этот параметр представляет приемлемый диапазон ошибок, генерируемых моделью, которые могут быть вызваны случайными редкими промахами. Это придает модели определенную гибкость для предотвращения ее переподгонки из-за промахов в тренировочном наборе.

Ядро выбирается посредством визуального просмотра распределения исследуемого набора данных. Каждый кластер в этом бимодальном распределении обладает гауссовыми характеристиками, поэтому можно предположить, что радиальная базисная функция (radial basis function – RBF) ядра должна стать хорошим выбором, принимая во внимание, что и значения функции Гаусса, и значения РБФ экспоненциально уменьшаются с удалением точек от центра.

Параметр `gamma` (**https://scikit-learn.org/stable/auto_examples/svm/plot_rbf_parameters.html**) используется для тонкой настройки РБФ ядра. Этот параметр определяет степень воздействия каждого отдельного элемента тренировочной выборки данных на итоговую модель. По умолчанию параметру `gamma` присваивается значение 0,5. При меньших значениях формируется более «гладкая» граница решения, которая, возможно, не способна правильно отобразить форму исследуемого набора данных. При бóльших значениях возникает вероятность переподгонки. В рассматриваемом примере выбрано меньшее значение `gamma` для предотвращения переподгонки из-за промахов, которые расположены слишком близко к границе решения.

Внимательно рассматривая итоговую модель, можно увидеть, что метод опорных векторов с одним классом вполне подходит для обработки этого строго бимодального набора данных и генерирует две моды кластеров нормальных данных, как показано на рис. 3.13. Здесь видны 16 неправильных классификаций, так что наличие промахов в тренировочных данных оказало некоторое воздействие на итоговую модель.

Попробуем еще раз провести тренировку модели, но уже на «чистых» нормальных данных и проверим, приведет ли это к каким-либо улучшениям. На рис. 3.14 показан результат второй тренировки.

На рис. 3.14 можно видеть, что на этот раз возникли только три ошибки классификации.

Метод опорных векторов с одним классом предлагает более гибкую методику подгонки распределения, используемого для обучения, к конкретному набо-

ру данных по сравнению с методикой устойчивой оценки ковариации. Но если приходится выбирать одну из этих методик в качестве механизма для системы выявления аномалий, то особое внимание необходимо уделить вероятности возникновения потенциальных промахов, которые могут искажать последующее выявление аномалий и приводить к постепенному снижению точности модели.

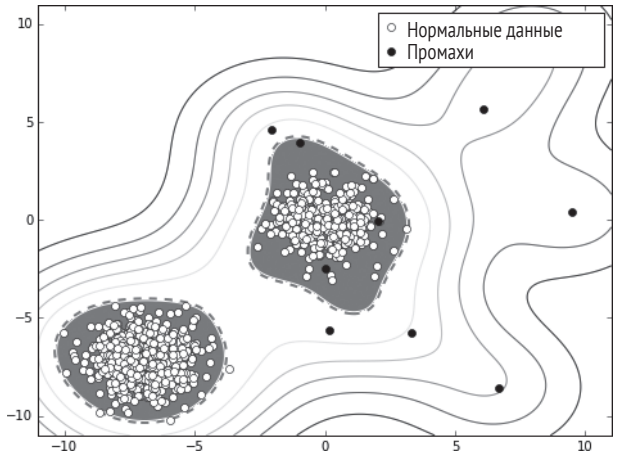


Рис. 3.13 ❖ Граница решения для метода опорных векторов с одним классом на бимодальных искусственно созданных данных с использованием для тренировки как промахов, так и нормальных данных

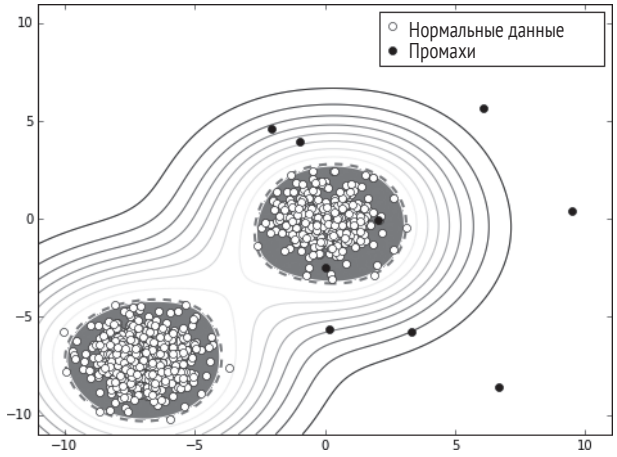


Рис. 3.14 ❖ Граница решения для метода опорных векторов с одним классом на бимодальных искусственно созданных данных с использованием для тренировки только нормальных данных

Изолирующие леса

Классификаторы на основе случайных лесов успешно работают в качестве механизмов выявления аномалий в многомерных наборах данных. Случайные леса – это алгоритмические деревья, а динамическая (потоковая) классификация

с применением структур данных в форме деревьев гораздо более эффективна по сравнению с моделями, использующими кластеры или вычисление функции расстояний. Число сравнений значений признаков, требуемых для классификации входящих точек данных, представляет высоту дерева (вертикальное расстояние между коренным узлом и конечным узлом-листом). Такая структура очень хорошо подходит для выявления аномалий в реальном времени на временно́м ряде данных.

Класс `sklearn.ensemble.IsolationForest` (**https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.IsolationForest.html**) помогает определить оценку аномалии в выборке, используя алгоритм изолирующего леса (Isolation Forest). Этот алгоритм тренирует модель посредством итеративного прохода по точкам данных в тренировочном наборе, случайно выбирая признак и разделяющее значение между максимальным и минимальным значениями этого признака (по всему набору данных). Алгоритм работает в контексте выявления аномалий с вычислением количества разделений, требуемых для изоляции одного элемента выборки, т. е. числа шагов-разделений признаков в наборе данных, необходимых для завершения пути в области, содержащей только один целевой элемент данных. Интуитивная суть этого метода состоит в том, что нормальные данные обладают бо́льшим сходством значений признаков, поэтому требуется больше операций разделения для их изоляции. С другой стороны, промахи должны изолироваться более просто и с меньшим числом разделений, так как некоторые значения их признаков с большей вероятностью существенно отличаются от значений признаков нормальных элементов. Измеряя «длину пути» рекурсивных разделений от корня дерева, мы получаем метрику, с помощью которой можно обосновать оценку аномалии для точек данных. Аномальные точки данных должны иметь более короткие пути, чем нормальные точки данных. В реализации библиотеки sklearn порговое значение для точек, которые должны считаться аномальными, определяется коэффициентом контаминации (contamination). Коэффициент контаминации 0,01 означает, что 1 % путей ветвления будет считаться аномалиями.

Рассмотрим практическое использование этого метода с применением алгоритма изолирующего леса к рассмотренному в предыдущих разделах «загрязненному» набору данных с распределением, отличным от распределения Гаусса (рис. 3.15)[1]:

```
from sklearn.ensemble import IsolationForest

rng = np.random.RandomState(99)
classifier = IsolationForest(max_samples=num_samples,
                             contamination=outlier_ratio,
                             random_state=rng)
classifier.fit(x)
y_pred = classifier.predict(x)
num_errors = sum(y_pred != labels)
print('Number of errors: {}'.format(num_errors))

> Number of errors: 8
```

[1] Полный код этого примера можно найти в Python Jupyter notebook chapter3/isolation-forest.ipynb (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter3/isolation-forest.ipynb**) в репозитории кода для этой книги.

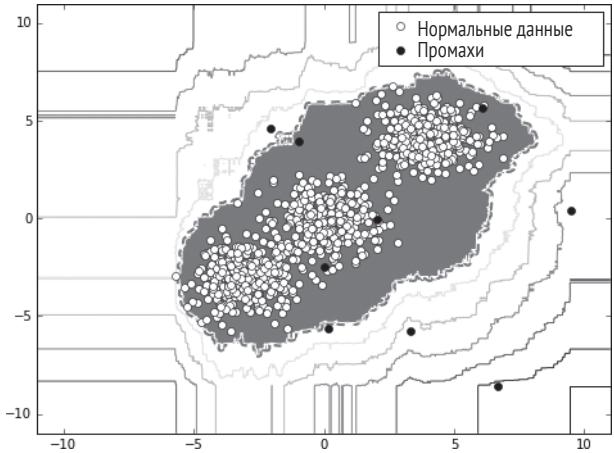


Рис. 3.15 ❖ Граница решения по алгоритму изолирующего леса для искуственно созданных данных с негауссовым распределением

Использование изолирующих лесов для выявления аномалий в динамических (потоковых) временны́х рядах очень похоже на использование метода опорных векторов с одним классом или на метод устойчивой оценки ковариации. Детектор аномалий просто выполняет разделения в дереве изолирующего леса и обновляет модель с помощью новых входящих точек (пока они не считаются аномалиями) во вновь изолируемых сегментах будущего пространства.

Важно отметить, что даже если тестирование/классификация выполняется эффективно, начальный этап обучения модели часто требует бóльших ресурсов и времени, чем другие методики выявления аномалий, рассмотренные выше. Для данных с весьма малой размерностью методика изолирующих деревьев для выявления аномалий может оказаться неподходящей из-за небольшого количества признаков, по которым можно выполнять разделение. Это ограничивает эффективность алгоритма.

Методы, основанные на плотности

Методы кластеризации, такие как алгоритм k-средних, известны по их использованию для классификации без учителя и регрессии. Можно воспользоваться похожими методами, основанными на плотности, в задачах выявления аномалий для идентификации промахов. Методы, основанные на плотности, хорошо подходят для наборов данных с высокой размерностью, которые могут вызывать затруднения при использовании других классов методик выявления аномалий. Некоторые методы, основанные на плотности, были специально адаптированы для решения задач выявления аномалий. Основная идея, заложенная в их основу, – формирование кластерного представления тренировочных данных в соответствии с предположением о том, что промахи или аномалии будут располагаться в областях с низкой плотностью этого кластерного представления. Такой подход обладает удобным свойством – устойчивостью к промахам в тренировочных данных, поскольку аномальные экземпляры с большой вероятностью также будут находиться в областях с низкой плотностью.

Даже если алгоритм k-ближайших соседей не является алгоритмом кластеризации, его принято считать методом, основанным на плотности, и достаточно часто применяемой методикой числовой оценки вероятности того, что точка данных является промахом. В сущности, алгоритм может давать оценку локальной плотности выборки для точки, измеряя расстояние от нее до k-го ближайшего соседа. Кроме того, можно воспользоваться кластеризацией методом k-средних для выявления аномалий похожим способом, используя расстояния между исследуемой точкой и центроидами как меру плотности выборки. Метод k-NN хорошо масштабируется для работы с большими наборами данных с использованием при этом k-мерных деревьев (k-d trees), которые могут значительно сократить время вычисления для наборов данных малых размерностей[1]. В этом разделе мы сосредоточим внимание на методике локального уровня выброса (промаха) (local outlier factor – LOF), представляющей классический метод машинного обучения, основанный на плотности, для выявления аномалий.

Локальный уровень выброса (промаха)

Методика локального уровня выброса (промаха), или просто LOF, является числовой оценкой аномалии, которую можно сгенерировать с помощью класса библиотеки scikit-learn sklearn.neighbors.LocalOutlierFactor (**https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.LocalOutlierFactor.html**). Подобно выше упомянутым методам выявления аномалий k-NN и k-средних, LOF классифицирует аномалии с использованием локальной плотности около некоторого элемента выборки. Локальная плотность точки данных – это концентрация других точек в ближайшей окрестности этой точки, где размер такой окрестности может быть определен либо как постоянное пороговое значение расстояния, либо по ближайшим n соседним точкам. LOF численно оценивает изолированность одной точки данных по отношению к ее ближайшим n соседям. Точки данных с существенно более низкой локальной плотностью по сравнению с ближайшими n соседями считаются аномалиями. Рассмотрим и выполним пример на том же «загрязненном» наборе данных с негауссовым распределением, который использовался в предыдущих разделах[2]:

```
from sklearn.neighbors import LocalOutlierFactor

classifier = LocalOutlierFactor(n_neighbors=100)
y_pred = classifier.fit_predict(x)
Z = classifier._decision_function(np.c_[xx.ravel(), yy.ravel()])
Z = Z.reshape(xx.shape)

num_errors = sum(y_pred != labels)
print('Number of errors: {}'.format(num_errors))

> Number of errors: 9
```

---

[1] *Alexandr Andoni and Piotr Indyk*. Nearest Neighbors in High-Dimensional Spaces: in Handbook of Discrete and Computational Geometry, 3rd ed., ed. Jacob E. Goodman, Joseph O'Rourke and Piotr Indyk (CRC Press).

[2] Полный код этого примера можно найти в Python Jupyter notebook chapter3/local-outlier-factor.ipynb (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter3/local-outlier-factor.ipynb**) в репозитории кода для этой книги.

На рис. 3.16 показан результат выполнения этого кода.

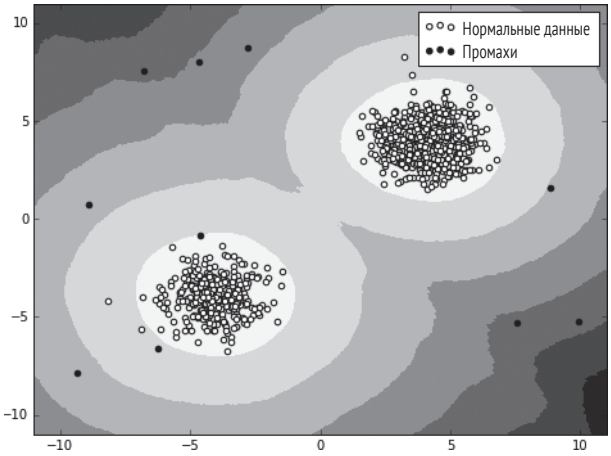


Рис. 3.16 ❖ Граница решения
по методу локального уровня выброса (промаха)
для бимодального искусственно созданного распределения данных

На рис. 3.16 можно видеть, что метод LOF работает хорошо даже при аномальных данных в тренировочном наборе и не слишком подвержен воздействию размерности данных. Пока набор данных сохраняет свойство меньшей локальной плотности в окрестности промахов по сравнению с их соседями в большинстве тренировочных признаков, метод LOF успешно формирует кластеры нормальных данных. Благодаря подходу, применяемому этим алгоритмом, появляется возможность выделить промахи в наборах данных с различной плотностью кластеров. Например, точка в разреженном кластере может иметь более длинное расстояние до своих ближайших соседей, чем другая точка в более плотном кластере (в другой области того же набора данных). Но поскольку сравнение плотностей проводится только для локальных соседей, для каждого кластера будут устанавливаться собственные условия по расстоянию, определяющие, содержит ли он промахи. Наконец, непараметрическая природа метода LOF означает, что его легко обобщить для применения к различным многочисленным измерениям, а также к числовым и непрерывным данным.

Краткие итоги

После анализа пяти категорий алгоритмов выявления аномалий должен стать очевидным тот факт, что существует достаточное количество методик машинного обучения, применимых к этой классической задаче майнинга данных. Выбор подходящего алгоритма иногда становится затруднительным и может потребовать выполнения нескольких циклов проб и ошибок. Но, пользуясь нашими указаниями и советами относительно того, какой класс алгоритмов наилучшим образом работает с определенными типами данных и для решения какого типа задач он пригоден в большей степени, вы окажетесь в более выгодной позиции, используя все преимущества машинного обучения для выявления аномалий.

Трудности применения машинного обучения для выявления аномалий

Одной из наиболее успешных областей применения машинного обучения являются рекомендательные системы (recommendation systems). Используя такие методики, как коллаборативная (или совместная) фильтрация, подобные системы способны определять скрытые предпочтения пользователей и действуют как механизм активной генерации запросов. А если предложена неправильная рекомендация? Если пользователю рекомендована несоответствующая продукция при просмотре сайта онлайновых покупок, то последствия незначительны. Принимая во внимание потенциально полезные рекомендации, пользователь просто игнорирует неинтересные ему рекомендации. Если ошибка возникла в алгоритме ранжирования личного поиска, то пользователь не найдет искомый товар, но это не столь ощутимая потеря.

В основу выявления аномалий заложена совершенно противоположная парадигма. Цена ошибок при выявлении вторжений и аномалий весьма высока. Неправильная классификация единственной аномалии может стать причиной критической уязвимости системы. Возникновение ложноположительных сигналов оказывает меньшее воздействие, но нежелательные ложноположительные случаи могут быстро снизить степень доверия к системе и даже привести к тому, что сигналы будут полностью игнорироваться. Из-за высокой цены ошибок классификации полностью автоматизированные непрерывно работающие системы выявления аномалий, основанные исключительно на методиках машинного обучения, встречаются крайне редко – почти всегда в рабочий цикл вовлечен человек для проверки правильности поступающих сигналов, прежде чем будут предприняты какие-либо ответные действия.

Семантический разрыв (semantic gap) – это реальная проблема машинного обучения во многих средах. По сравнению со статическим набором правил или эвристик, иногда трудно объяснить, почему некоторое событие было помечено как аномалия. Это приводит к длительным циклам расследования инцидента. На практике интерпретируемость и объяснимость результатов часто не менее важны, чем точность. Особенно для систем выявления аномалий, которые постоянно модифицируют свои модели решений со временем, чрезвычайно важно рациональное планирование и распределение ресурсов в компонентах системы, которые могут формировать удобные для человека описания сигналов, генерируемых системой машинного обучения. Например, если сигнал подан системой выявления промахов на основе метода опорных векторов с одним классом с использованием скрытой комбинации признаков, выбираемых с помощью методик сокращения размерности, то для человека может быть затруднительным определение соответствующего сочетания явных сигналов системы. Учитывая непрозрачность многих процессов машинного обучения, во всех возможных случаях чрезвычайно полезно генерировать объяснения решений, принятых моделью.

Разработка схемы правильных оценок для систем выявления аномалий может оказаться даже более сложной задачей, чем создание самой системы. Поскольку при выявлении аномалий для временны́х рядов данных допускается вероятность того, что входные данные никогда не появлялись в прошлом, не

существует полноценного способа оценки системы из-за огромного количества возможностей возникновения разнообразных аномалий, которые могут встретиться на практике.

Опытные взломщики могут (и будут) тратить время и усилия на то, чтобы обойти системы выявления аномалий, если в перспективе имеется заслуживающий внимания результат. Влияние конкурирующего внедрения систем и алгоритмов машинного обучения вполне реально и является необходимым условием для развертывания систем в потенциально опасной среде. В главе 8 более подробно рассматривается конкурирующее машинное обучение, но любая безопасная система машинного обучения должна иметь встроенную защиту от искажений и мошеннических действий. Такие методики защиты также будут рассматриваться в главе 8.

Ответная реакция и ослабление воздействия

Что происходит после получения сигнала об аномалии? Ответная реакция на отрицательное событие и меры по ослаблению его воздействия являются практическими областями, заслуживающими собственных исследований и публикаций. Но здесь у нас нет возможности полностью описать все связанные с этим нюансы и сложности. Тем не менее мы рассмотрим, как машинное обучение может быть введено в обычные потоки операций по обеспечению безопасности для улучшения работоспособности и снижения доли участия человека.

Простые сигналы об аномалиях могут приходить в форме сообщений электронной почты или оповещений по мобильной связи. В большинстве случаев организации, сопровождающие множество разнообразных систем выявления аномалий и мониторинга безопасности, по достоинству оценивают полезность объединения сигналов из многих источников в одной платформе под названием система Security Information and Event Management (SIEM). Системы SIEM могут помочь в управлении выходными данными систем обеспечения безопасности, разделенных на сегменты, которые могут быстро увеличиваться в размерах. Системы SIEM также устанавливают соответствие между сигналами, сгенерированными различными системами, чтобы помочь аналитикам объединить наблюдения из самых разнообразных систем обеспечения безопасности и выявления нарушений.

Наличие единой локации формирования отчетов и обработки сигналов также создает существенное различие в оценках генерируемых сигналов от систем защиты. Сигналы об угрозах безопасности нередко могут становиться причиной активизации ответных действий для различных подразделений организации, а не только группы обеспечения безопасности или даже инженерной группы. Часто улучшения системы обеспечения безопасности любой организации требуют правильной координации усилий по общему управлению всеми подразделениями, многие из которых не всегда обладают знаниями о внутреннем уровне операций по обеспечению безопасности и соответствующим опытом. Наличие платформы, которая может помочь в составлении отчетов и понятных, удобных для чтения человеком объяснений случаев нарушения системы безопасности, становится чрезвычайно ценным инструментом при необходимом обмене информацией по вопросам безопасности между организацией и внешними заинтересованными сторонами.

Ответная реакция на происшествие обычно требует привлечения человека в конечном пункте приема сигналов от системы безопасности и выполнения вручную операций по исследованию, проверке и передаче этих сигналов на следующие этапы обработки. Ответная реакция на подозрительное событие часто связывается с цифровой криминалистикой и, как следствие, с областью цифровой криминалистики и ответных судебных действий (digital forensics and incident response – DFIR), включающей в себя широкий спектр действий, которые аналитики по проблемам безопасности должны выполнять для классификации сигналов, для сбора доказательств при расследованиях, для проверки аутентичности собранных данных и для предоставления информации в формате, удобном для обработки на следующих инстанциях. Несмотря на то что в других областях обеспечения безопасности постоянно повышается уровень автоматизации, сфера ответных судебных действий неизменно остается процессом, выполняемым вручную. Например, существуют инструментальные средства, помогающие инспектировать бинарные файлы и просматривать дампы оперативной памяти, но реальной замены человеку, формулирующему предположения относительно вероятных действий атакующих и их намерений и целей по использованию взломанного хоста, пока еще не существует.

При этом ответная реакция на происшествие с машинной поддержкой выглядит многообещающим направлением. Машинное обучение способно эффективно обрабатывать крупные наборы данных для выявления шаблонов и аномалий, в то время как людям-аналитикам предоставляется возможность делать обоснованные предположения и выполнять более сложные задачи, требующие глубоких ситуативных практических (т. е. связанных с конкретной областью деятельности) знаний. Объединение этих двух взаимодополняющих направлений приложения усилий может повысить эффективность операций, выполняемых как ответная реакция на определенное событие.

Ослабление, или смягчение, угрозы (threat mitigation) – это процесс формирования ответной реакции на вторжения и атаки, а также предотвращения подобных действий. Первой реакцией на сигнал о вторжении может быть пресечение этого опасного действия в его начальной стадии и организация защиты от дальнейшей деятельности взломщика. Но такая реакция препятствует сбору более подробной информации о возможностях, намерениях и исходных позициях атакующего. В среде, где атакующие могут быстро создать опорный пункт для многократной реализации своих стратегий ухода от обнаружения, их выдворение или блокировка может привести к нежелательным последствиям. Немедленная ответная реакция на атакующих может дать им информацию о том, каким образом они были обнаружены, что позволит им постепенно найти такое место в системе, где их будет весьма трудно обнаружить. Незаметное наблюдение за атакующими с ограничением их возможностей причинения вреда является более эффективной оперативной мерой, предоставляющей защищающейся стороне больше времени для разработки долговременной стратегии, способной обеспечить надежную защиту от подобных атак.

Скрытая блокировка (stealth banning), или теневая блокировка (shadow banning) (она же hell banning или ghost banning), – практическая методика, специально предназначенная для социальных сетей и платформ онлайновых сообществ с целью избирательной блокировки оскорбительного контента или спама. Основ-

ная цель – не позволить нарушителям немедленно сформировать цикл обратной связи. Платформа скрытой блокировки создает искусственную среду, доступную для атакующих после их обнаружения. В этой среде атакующие рассматриваются так же, как на обычной платформе, поэтому они продолжают считать, что их действия правомерны, хотя в действительности каждый, находящийся под действием скрытой блокировки, не может создавать какие-либо побочные эффекты и абсолютно невидим для других пользователей или системных компонентов (т. е. атакующему объявлен своеобразный бойкот).

Практические аспекты проектирования систем

При проектировании и реализации систем машинного обучения для обеспечения безопасности следует принять во внимание ряд практических решений по проектированию системы, позволяющих улучшить точность классификации.

Оптимизация объяснимости

Как было отмечено выше, семантический разрыв в объяснимости сигнала тревоги является одним из самых значительных препятствий, которые затрудняют работу детекторов аномалий, использующих машинное обучение. Для многих практических приложений машинного обучения высоко ценятся объяснения результатов. Тем не менее правильное объяснение в методах машинного обучения является областью активных исследований, которая пока еще не дает определенных ответов на поставленные вопросы.

Простые классификаторы с использованием машинного обучения и даже механизмы классификации без применения машинного обучения дают достаточно понятные прогнозы. Например, модель линейной регрессии на двумерном наборе данных генерирует вполне объяснимые результаты, но неспособна обучаться на более сложных и многозначных признаках. Более сложные модели машинного обучения, такие как нейронные сети, классификаторы на основе случайных лесов и методики ансамблирования, лучше подходят для работы с реальными данными, но представляют собой совершенно черные ящики – процессы принятия решений полностью скрыты от внешнего наблюдателя. Однако существуют способы решения и этой проблемы, которые могут применяться к прогнозам машинного обучения, сложным для объяснения, подтверждая тот факт, что объяснимость в действительности не противоречит точности прогнозов[1]. Наличие внешней системы, генерирующей простые, удобные для чтения человеком объяснения решений, принятых классификатором как черным ящиком, удовлетворяет условиям объяснимости результата[2], даже если эти объяснения не описывают действительные условия принятия решения системы машинного обучения. Такая внешняя система анализирует любые выходные данные системы машинного обучения и выполняет контекстный анализ этих данных для формулирования наиболее вероятных причин возникновения конкретного сигнала об опасности.

---

[1] В главе 7 более подробно рассматривается решение проблем объяснимости в машинном обучении.

[2] *Ryan Turner*. A Model Explanation System. Black Box Learning and Inference NIPS Workshop (2015).

Производительность и масштабируемость в потоковых приложениях реального времени

Многие приложения по выявлению аномалий в контексте обеспечения безопасности требуют наличия системы, способной обрабатывать в реальном времени потоковые запросы на классификацию и правильно учитывать смещение трендов в данных, происходящие во времени. Но, в отличие от специализированных процессов машинного обучения, здесь точность классификации не является единственной оптимизируемой метрикой. Несмотря на то что такие процессы могут выдавать несоответствующие (или несвоевременные) результаты классификации, некоторые алгоритмы менее требовательны ко времени и ресурсам, чем другие, поэтому могут стать оптимальным вариантом выбора при проектировании систем для сред с критически ограниченными ресурсами (например, для реализации методов машинного обучения на мобильных устройствах или во встроенных системах).

В сфере ИТ распараллеливание является классическим ответом на возникающие проблемы производительности. Алгоритмы машинного обучения с возможностью параллельного выполнения и/или распределенной работы в рабочих средах MapReduce, таких как Apache Spark (Streaming) (**http://spark.apache.org/streaming/**), – неплохой способ значительного улучения производительности (до порядка). В системах, проектируемых для реального практического применения, следует помнить о том, что некоторые алгоритмы машинного обучения весьма трудно перевести в параллельный режим, поскольку они требуют обмена информацией между узлами (например, простые алгоритмы кластеризации). Использование библиотек машинного обучения с поддержкой распределенного режима, например Apache Spark MLib (**http://spark.apache.org/mllib/**), может помочь избежать трудностей при реализации и оптимизации распределенных систем машинного обучения. Более подробно применение таких программных сред и инструментальных средств будет рассматриваться в главе 7.

Удобство сопровождения систем выявления аномалий

Долговечность и полезность систем машинного обучения определяется не только точностью и эффективностью, но также объяснимостью, удобством сопровождения и простотой конфигурирования программного обеспечения. Проектирование модульной системы, которая обеспечивает оперативную замену, удаление и ввод новых реализаций своих компонентов, чрезвычайно важно в постоянно изменяющихся средах. Природа данных постоянно изменяется, и самая современная высокоэффективная модель машинного обучения может оказаться не соответствующей требованиям через полгода эксплуатации. Если система выявления аномалий спроектирована и реализована исходя из предусловия использования метода подгонки эллиптической огибающей кривой, то в будущем очень трудно будет заменить этот алгоритм, например, на изолирующие леса. Гибкая конфигурация как параметров самой системы, так и параметров применяемого алгоритма важна по одной и той же причине. Если тонкая настройка параметров модели требует перекомпиляции бинарных файлов, то такую систему нельзя считать удобно конфигурируемой.

Внедрение обратной связи с человеком

Наличие цикла обратной связи в системе выявления аномалий может способствовать весьма высокой адаптируемости системы. Если аналитики в области безопасности получают возможность передать отчет о ложноположительных и ложноотрицательных событиях непосредственно в систему, которая настраивает параметры модели на основе этой обратной связи, то удобство сопровождения и гибкость такой системы могут существенно улучшиться. Но в ненадежных средах непосредственное внедрение цикла обратной связи с человеком в тренировочную модель может приводить к отрицательным эффектам.

Снижение воздействий состязательности

Как было отмечено выше, во враждебной среде системы обеспечения безопасности на основе методов машинного обучения почти наверняка подвергаются атакам. Нападающие на такие системы в основном пользуются одним из двух классов методик для достижения своих целей. Если система постоянно обучается на входных данных, а оперативная разметка выполняется в цикле обратной связи, поддерживаемом пользователями (онлайновая модель обучения), то атакующий может «заразить» (poison) модель, внедряя преднамеренно некорректный трафик с помехами (chaff) для искажения границ решения, формируемых классификаторами. Кроме того, атакующие могут уклоняться (evade) от классификаторов с помощью состязательных примеров (adversarial examples), специально адаптированных для обмана конкретных моделей и реализаций. Важно подготовить специализированные процессы для безусловного предотвращения этих векторов угроз и проникновения в защищаемую систему. Особенно опасен подход к проектированию системы, при котором любые пользовательские входные данные безрассудно принимаются для обновления модели. В онлайновой модели обучения, рассматривающей любые входные данные как объект преобразования в тренировочные данные для модели, важно выявлять попытки «заражения» системы. Использование надежных статистических данных, устойчивых к искажениям и попыткам зондирования, – еще один способ снижения эффективности атаки. Подготовка и сопровождение тестовых наборов данных и эвристик, по которым периодически выполняются проверки отклонений от нормы входных данных, границы решений модели или результатов классификации, также может быть весьма полезным. Проблемы состязательности и их решения более подробно будут рассматриваться в главе 8.

Резюме

Выявление аномалий – это область, в которой методики машинного обучения продемонстрировали высокую эффективность. Прежде чем углубляться в изучение сложных алгоритмов и статистических моделей, следует тщательно обдумать задачу, которую вы пытаетесь решить, и доступные данные. Решение по созданию успешно работающей системы выявления аномалий может заключаться не в использовании более продвинутого алгоритма, а в генерации более полного и наглядного (объясняемого) набора входных данных. Из-за чрезвычайно широкого спектра потенциальных угроз, требующих смягчения их негативного воздействия, сложность систем обеспечения безопасности имеет тенденцию к неконтролируемому росту. При создании или усовершенствовании систем выявления аномалий необходимо всегда сохранять простоту системы как цель с наивысшим приоритетом.

Глава 4

Анализ вредоносного программного обеспечения

Когда в 2010 г. центрифуги с воздушным разрядным зазором на горно-обогатительном комбинате по обработке ядерного сырья (урана) в иранском городе Нетенз (Natanz) неожиданно вышли из строя, никто точно не знал, кто несет ответственность за эту аварию. Червь Stuxnet стал в то время одним из самых сенсационных событий в международной кибервойне и демонстрацией изменения ситуации и резко расширившихся разрушительных возможностей вредоносного программного обеспечения при экстремально дальнем расстоянии его действия. Этот экземпляр вредоносного программного обеспечения беспорядочно распространял себя по всему миру, как только находил подходящую промышленную компьютерную систему, которые являлись его целью. По сообщениям, Stuxnet в итоге внедрился в десятки тысяч компьютеров под управлением ОС Windows, сохраняя при этом свое скрытое состояние, в результате чего была нарушена работа одной пятой всех ядерных установок в Иране. По некоторым утверждениям, главной целью этой атаки было создание препятствий для государственной программы по созданию ядерного оружия.

Анализ вредоносного программного обеспечения (ПО) (malware analysis) – это область исследования функциональных возможностей, целей, происхождения и потенциального воздействия вредоносного ПО. Эта задача обычно выполняется в основном вручную и связана со значительными трудозатратами. Для ее решения требуются аналитики с экспертным уровнем знаний о внутренней организации ПО и реверс-инженерии. Даталогия и машинное обучение обладают многообещающими перспективами в сфере автоматизации некоторых этапов анализа вредоносного ПО, но их методики главным образом остаются связанными с процессом извлечения наиболее важных признаков из данных. Это весьма непростая задача, для решения которой также требуются эксперты-практики со специальным набором знаний и умений.

В этой главе мы не будем рассматривать методы статистического обучения[1]. Вместо этого мы обсудим один из наиболее важных, но часто недооцениваемых этапов машинного обучения: конструирование признаков (feature engineering). В этой главе мы пытаемся объяснить поведение и внутренние принципы работы вредоносных бинарных выполняемых файлов. Особое внимание будет уделено

[1] Методы статистического обучения, такие как классификация, кластеризация и выявление аномалий, подробно рассматриваются в главах 2, 3 и 5 данной книги.

решению задачи анализа и классификации вредоносного ПО с позиции науки о данных, т. е. описанию того, как нужно правильно извлекать полезную информацию из бинарных файлов.

Эта глава разделена на две части, поскольку для плодотворного обсуждения практического применения методики конструирования признаков к вредоносному ПО необходим определенный объем дополнительных знаний. В первой части «Что такое вредоносное программное обеспечение» приводится обоснование способов классификации вредоносного ПО, организация и структура вредоносного ПО, механизмы выполнения программ и типовые шаблоны поведения различных классов вредоносного ПО. Эта информация формирует основу для изучения второй части «Генерация признаков», в которой рассматриваются специализированные методики извлечения и конструирования признаков из бинарных данных различных форматов[1] для использования в методиках даталогии и машинного обучения.

Что такое вредоносное программное обеспечение

Исходный код проходит несколько этапов обработки, прежде чем начать свое выполнение в качестве программы на компьютере. Хорошее знание этих этапов чрезвычайно важно для любого аналитика вредоносного ПО. Существует множество различных типов вредоносного ПО и так же, как и различные типы обычного ПО, каждый тип потенциально может быть написан на одном из многочисленных языков программирования с учетом различий целевых сред выполнения и разнообразных требований к условиям выполнения. Если есть доступ к коду высокого уровня (C/C++, Java или Python), то относительно просто можно определить, что делает конкретная программа и как получить набор ее внутренних характеристик, т. е. профиль программы, ее поведение. Но маловероятно, что удастся с легкостью получить доступ к коду высокого уровня, использованному для создания вредоносного ПО. Большинство подобных программ перехватывается и накапливается «в полевых условиях», отлавливается в ловушках-песочницах, продается на нелегальных форумах, кроме того, их можно обнаружить на компьютерах невольных жертв. В упакованном и развернутом состоянии большинство вредоносных программ существует в виде бинарных файлов, которые чаще всего человек не может прочитать, поскольку они предназначены для непосредственного выполнения на компьютере. Таким образом, профилирование характеристик и поведения вредоносной программы становится процессом реверс-инженерии, цель которого – понять, что делает эта программа, как если бы у нас имелся в распоряжении исходный код на языке высокого уровня.

---

[1] В этой главе при использовании термина «бинарные данные» (binary data) подразумевается формат представления данных, состоящий исключительно из нулей и единиц. Каждый отдельный знак (цифра) 0/1 называется битом (bit), каждая непрерывная последовательность из восьми битов называется байтом (byte). В современных компьютерных системах бинарные файлы присутствуют всегда, а программные функции выполняют преобразование представления на уровне битов/байтов в абстрактную информацию более высокого уровня для дальнейшей интерпретации другими компонентами ПО (инструкции ассемблера, распакованные файлы и т. п.) или для вывода в интерфейсе пользователя (текст, изображения, звук и т. д.).

Бинарные файлы по своей природе непонятны, что чрезвычайно затрудняет любые попытки извлечь из них какую-либо информацию. Без знания контекста интерпретации, стандартов кодирования и алгоритма декодирования сами по себе бинарные данные фактически бессмысленны. Как отмечалось в предыдущих главах, система машинного обучения хороша лишь настолько, насколько качественными являются ее входные данные. Исходные бинарные данные, даже в большей степени, чем другие формы входных данных, требуют составления плана сбора данных, их очистки и проверки (валидации), прежде чем передать их в алгоритм машинного обучения. Предварительная обработка таких исходных бинарных данных важна для выбора их оптимального формата и представления при передаче в алгоритм обучения.

В этой книге весь процесс сбора и преобразования данных в формат, подходящий для передачи как входных данных в алгоритмы, в широком смысле обозначается термином «конструирование признаков» (feature engineering). Извлечение признаков (feature extraction) – термин, который здесь используется для описания процесса извлечения признаков из необработанных исходных бинарных данных. Например, если необходимо классифицировать музыкальные файлы в формате WAV[1] по различным жанрам (например, классика, рок, поп, джаз и т. п.), то исходными данными должно быть содержимое WAV-файлов. Самым очевидным преобразованием содержимого каждого WAV-файла во входные данные для алгоритма машинного обучения является бинарное представление такого файла на уровне битов. Тем не менее подобную форму нельзя назвать самым удобным и эффективным представлением звуковых (музыкальных) файлов. Вместо этого можно выполнить конструирование признаков из необработанных входных данных для генерации других форм представлений. Например, пропустить исходные данные через программу анализа звука, чтобы извлечь такие признаки, как минимальная, максимальная и средняя амплитуда и частота. Более изощренные программы анализа способны извлекать такие признаки, как ритм (число тактов в минуту), тональность произведения и более тонкие полифонические характеристики музыки. Разумеется, такие характеристики могут помочь составить более полное представление о каждом музыкальном произведении, позволяя классификатору на основе машинного обучения распознавать различия в темпе, ритме и тональных характеристиках для образцов разных музыкальных жанров.

Для идентификации и извлечения правильных признаков, необходимых для выполнения анализа безопасности компьютерных бинарных файлов, требуется глубокое понимание внутренних принципов работы программного обеспечения. Эта область называется реверс-инженерией ПО (software reverse engineering) и представляет собой процесс извлечения информации и практических знаний о внутреннем функционировании программ. Это необходимо для полного понимания, какими свойствами обладает ПО, как оно работает и какие недостатки имеет. С помощью реверс-инженерии бинарного файла можно понять его функциональность, предназначение, а иногда даже его происхождение. Реверс-инженерия – это особый навык, требующий длительной тренировки и практики, но эта глава не является полноценным руководствам по реверс-инженерии, для

[1] WAV (или WAVE) – это стандартный формат звуковых файлов для хранения аудиопотока битов на компьютерах.

этого существуют специализированные книги и прочие учебные материалы[1]. Основная цель данной главы – объяснить основы методики генерации признаков с использованием принципов реверс-инженерии. Хорошо понимая, как работают компоненты ПО, и идентифицируя особенные свойства, присущие их функциональности, можно более успешно конструировать признаки, которые позволят алгоритмам машинного обучения давать более точные прогнозы.

Вредоносное ПО может быть встроено в бинарные файлы разнообразных форматов, работа которых совершенно отличается друг от друга. Например, PE-файлы в ОС Windows (переносимые выполняемые файлы – Portable Executables, с расширениями *.exe*, *.dll*, *.efi* и т. д.), ELF-файлы в Unix-системах (Executable and Linkable Format) и APK-файлы в ОС Android (формат Android Package Kit с расширением *.apk* и др.) имеют совершенно различную внутреннюю структуру файлов и требуют разного контекста выполнения. Вполне естественно, что для анализа каждого класса выполняемых файлов специальные дополнительные требования также абсолютно различны. Кроме того, необходимо учесть, что вредоносное ПО может существовать и в формах, отличающихся от отдельных бинарных выполняемых файлов. Широко распространены вредоносные компоненты, внедряющиеся в файлы документов, например с расширениями *.doc*, *.pdf* и *.rtf*, и использующие макро (macros)[2] и динамически выполняемые элементы в структуре документа, чтобы выполнить вредительские действия. Вредоносное ПО также может быть представлено в форме расширений и динамически подключаемых компонентов (plug-ins) для распространенных программных платформ, таких как веб-браузеры и комплексные рабочие веб-среды. Мы не будем подробно обсуждать каждый из этих форматов, а только лишь рассмотрим наиболее важные различия между ними. Немного больше внимания будет уделено формату Android APK как практическому примеру для ваших собственных исследований и разработок в области анализа данных вредоносного ПО.

Классификация вредоносного программного обеспечения

Прежде чем приступить к препарированию бинарных файлов, необходимо ознакомиться с некоторыми определениями. Группы классификации вредоносного ПО объединяют отдельные экземпляры на основе общих свойств. Можно классифицировать вредоносное ПО разнообразными способами в зависимости от конкретной задачи и поставленной цели. Например, группа обеспечения безопасности может классифицировать вредоносные программы по степени их опасности и функциональности для более точного определения степеней угрозы для организации. Группы оперативной реакции могут классифицировать вредоносные программы по потенциальному размеру ущерба и по вектору внедрения, чтобы

---

[1] Например, Practical Malware Analysis by Michael Sikorski and Andrew Honig (No Starch Press) и Michael Hale Ligh et al.'s Malware Analyst's Cookbook (Wiley).

[2] Макро (macro) – это набор команд для автоматического выполнения часто повторяющихся задач в среде приложений, например в Microsoft Word или Excel. Вредоносные макро получили широкое распространение в 1990-е годы, используя для своих целей возможности автоматизации популярных пользовательских программ для запуска вредоносного кода на компьютере жертвы. В последние годы наблюдается возобновление активности вредносных макро, часто являющихся элементами кампаний, организованных группами социальной инженерии с целью широкого распространения.

разработать стратегии устранения и/или снижения уровня ущерба. Исследователи вредоносного ПО могут распределять экземпляры по категориям на основе их происхождения и авторства, чтобы лучше понять генеалогию и цели.

Для обобщенного анализа вредоносного ПО распространенной промышленной практической методикой является группировка экземпляров по семействам (family). Этот термин используется аналитиками вредоносного ПО и позволяет прослеживать авторство, коррелировать информацию и идентифицировать новые варианты обнаруженных вредоносных программ. Экземпляры вредоносного ПО из одного семейства могут иметь одинаковый код, возможности, авторство, функциональные характеристики, цели и/или исходные предпосылки. Широко известным примером семейства вредоносного ПО является Conficker – червь, предназначенный для внедрения в ОС Microsoft Windows. Несмотря на то что существует множество вариаций червя Conficker с различным кодом, авторами и поведением, определенные характеристики всех этих червей позволяют причислить их к одному семейству, тем самым обозначая их происхождение от одного ранее известного предшественника. Например, все черви Conficker используют уязвимости ОС Windows и предпринимают атаки по словарям для подбора пароля к учетной записи администратора. После этого черви устанавливают скрытое ПО, чтобы использовать зараженный хост в деятельности ботнета.

Различия между экземплярами вредоносного ПО из одного семейства могут определяться по разным компиляторам, используемым для компиляции исходного кода, или по секциям кода, добавляемым и/или удаляемым для изменения функциональности самой вредоносной программы. Экземпляры вредоносных программ, которые со временем эволюционируют в ответ на изменения стратегий их выявления и нейтрализации, зачастую также демонстрируют сходство между старой и новой версиями, позволяя аналитикам прослеживать развитие семейства вредоносного ПО. В любом случае распределение по семействам вредоносного ПО несомненно является трудной задачей, которая может давать различные результаты в зависимости от классификационных определений и методов, используемых аналитиками.

Классификация вредоносного ПО также может быть обобщена при помощи включения в нее бинарных файлов, не являющихся вредоносными. Этот тип классификации используется для определения факта принадлежности того или иного компонента ПО к группе вредоносных программ. Рассматривая произвольный бинарный файл, необходимо знать вероятность того, что ему можно доверять, и выполнять его в защищаемой среде. Это основная цель антивирусного ПО, представляющая собой особенно важную задачу для специалистов по компьютерной безопасности, поскольку данное знание может помочь предотвратить распространение вредоносного ПО в организации. Обычно решение такой задачи основано на сравнении сигнатур: имея достаточный набор свойств и образцов поведения ранее обнаруженного и исследованного вредоносного ПО, можно сравнивать новые появляющиеся в системе бинарные файлы с этим набором данных, чтобы определить, нет ли совпадений с какими-либо признаками, выявленными ранее.

Метод сравнения сигнатур хорошо работает до тех пор, пока авторы вредоносных программ не вносят существенных изменений в свойства и поведение своих программ, чтобы избежать обнаружения. Характерные свойства и образцы по-

ведения сохраняют благоприятный баланс и обеспечивают стабильность сигнала (т. е. все экземпляры вредоносной программы, принадлежащие к конкретному семейству, являются причиной именно этого сигнала) и его характерные особенности (т. е. «добропорядочные» бинарные файлы не демонстрируют тех свойств или поведения, которые могли бы стать причиной их неправильной классификации как вредоносного ПО). Но авторы вредоносных программ неуклонно стремятся непрерывно изменять их свойства и поведение, чтобы избежать обнаружения.

Метаморфные или полиморфные[1] вирусы и черви используют статические и динамические методики маскировки для изменения характеристик своего кода, поведения и свойств, применяемые для алгоритмов генерации сигнатур в механизмах идентификации вредоносного ПО. Такой уровень изобретательности при создании вредоносной программы раньше встречался редко, но в настоящее время становится все более распространенным из-за неизменно успешного создания помех и препятствий для механизмов синтаксических сигнатур вредоносного ПО. Механизмы синтаксических сигнатур продолжают погоню за постоянно сужающимся набором статических сигналов, которым авторы вредоносного ПО пренебрегают или не могут изменить коренным образом.

Машинное обучение в классификации вредоносного программного обеспечения

Наука о данных и машинное обучение могут помочь при устранении некоторых типов проблем, создаваемых современным вредоносным ПО, главным образом благодаря трем характеристикам, которые обеспечивают преимущество над методикой статического сравнения сигнатур:

- нечеткое сравнение (fuzzy matching): алгоритмы машинного обучения могут определять сходство между двумя и более объектами с помощью метрики расстояния. Механизмы оценки сходства, которые раньше генерировали бинарный результат – совпадение или несовпадение, – в настоящее время способны выводить действительное число в интервале от 0 до 1, обозначающее оценку (степень) достоверности (confidence score), соответствующую выводу алгоритма относительно того, насколько вероятным можно считать сходство двух объектов, т. е. принадлежность их к одному классу. По аналогии с интуитивным примером применения методов кластеризации экземпляры данных, отображаемые в векторное пространство признаков, могут быть сгруппированы на основе относительных расстояний между ними. Точки, близкие друг к другу, могут считаться весьма схожими, тогда как точки, удаленные друг от друга, должны считаться абсолютно несхожими.

  Такая возможность определения приблизительных совпадений между объектами чрезвычайно полезна при классификации вредоносного ПО, разли-

[1] Существует тонкое различие между метаморфизмом и полиморфизмом вредоносного ПО. Полиморфные вредоносные программы обычно содержат две секции кода: ядро логики, выполняющее собственно заражение, и внешнюю объемлющую секцию, которая использует разнообразные формы зашифрования и расшифрования для маскировки инфицирующего кода. Метаморфные вредоносные программы внедряют, перекомпоновывают, подменяют версии реализации, добавляют и удаляют фрагменты своего кода. Поскольку логика заражения не изменяется на каждой стадии эволюции вредоносной программы, полиморфные программы проще выявлять, чем метаморфные программы.

чия в свойствах которого приводят в замешательство методы статического сравнения сигнатур;

- автоматизированный выбор свойств: автоматическое определение веса (значимости) признака и выбор самых значимых признаков являются главными аспектами машинного обучения, которые помогают осуществить классификацию вредоносного ПО. На основе статистических свойств тренировочного набора данных можно ранжировать признаки по их относительной важности при определении отличия экземпляра, принадлежащего классу A, от другого экземпляра, принадлежащего классу B. Кроме того, появляется возможность объединения в группу двух экземпляров класса A. Ранее классификация вредоносного ПО обычно представляла собой задачу, выполняемую преимущественно вручную, требуя для этого большого объема экспертных знаний о том, как работают вредоносные программы и какие свойства используются в механизме их классификации. Некоторые алгоритмы снижения размерности и выбора признаков способны обнаруживать скрытые свойства экземпляров, которые чрезвычайно трудно выявить даже эксперту в этой области.

  Методы машинного обучения освобождают аналитиков вредоносного ПО от некоторых трудоемких операций по определению значимости каждого признака. Позволяя самим данным выявлять и определять набор признаков для использования в схеме классификации, аналитики получают возможность сосредоточиться на конструировании признаков и предоставить алгоритму более полный и более подробно описанный набор данных, что способствует повышению эффективности;
- адаптируемость: вечное противостояние между создателями вредоносного ПО и защитниками систем приводит к непрерывному изменению типов и шаблонов предпринимаемых атак. Как и при разработке обычного ПО, вредоносное ПО развивается и улучшается со временем, поскольку его авторы добавляют функциональные возможности и исправляют ошибки. Кроме того, как было отмечено выше, авторы вредоносного ПО всегда стремятся к непрерывным усовершенствованиям, изменяя поведение своих программ, чтобы избежать обнаружения. С помощью методики нечеткого сравнения и управляемого данными процесса выбора признаков системы классификации вредоносного ПО, реализованные на основе машинного обучения, способны адаптироваться к изменениям исходных условий (входных данных) и отслеживать развитие вредоносного ПО во времени.

  Например, экземпляры из семейства червей Conficker с 2008 по 2010 г. могли проявляться в совершенно различных вариациях и демонстрировать разнообразное поведение. Адаптивная система классификации, которая непрерывно отслеживала и выявляла постепенные изменения в экземплярах этого семейства, научилась находить свойства, соответствующие не только ранним выборкам данных, но и модифицированным экземплярам из того же семейства.

Определение авторства вредоносных программ, возможно, не является первостепенно важной частью задачи классификации, но полное понимание целей, намерений и источников атак может помочь защищающимся разработать более дальновидные стратегии противодействия, которые предотвратят настойчивые попытки взломщиков проникнуть в систему.

Машинное обучение может помочь существенно снизить объем ручной работы и уровень экспертных знаний, требуемый для классификации вредоносного ПО. Позволяя данным и алгоритмам управлять решениями, требующими четкого определения отношений между большим количеством экземпляров выборки, мы получаем более точные результаты, чем человек, выполняющий ту же работу. Поиск шаблонов (образцов) и схожести данных – это самая сильная сторона алгоритмов машинного обучения, но некоторые этапы задачи все еще требуют привлечения к работе человека. Генерация хорошо описанных наборов данных в формате, помогающем алгоритмам при решении задач обучения и классификации, – для этого необходим специалист по работе с данными, который глубоко понимает принципы работы как вредоносного ПО, так и алгоритмов машинного обучения.

Вредоносное программное обучение: что скрывается внутри

При генерации хорошо описанного набора данных для классификации вредоносного ПО необходимо понимать, как работает этот тип программ. Это, в свою очередь, требует рассмотрения соответствующих экономических и организационных аспектов, общих типов вредоносного ПО и обобщенной схемы процессов выполнения программ в современных компьютерных средах.

Экономические и организационные аспекты вредоносного программного обеспечения

В разделе «Экономическая подоплека кибератак» главы 1 было отмечено, что экономика и организация вредоносного ПО весьма энергичны, но неустойчивы из-за неизбежного дисбаланса между стоимостью (затратами) и выгодами распространяемых вредоносных программ. Рассматривая эту тему с точки зрения экономики, легко понять, почему вредоносное ПО столь широко распространено. От распространителей вредоносных программ требуются лишь минимальные трудозатраты или небольшая денежная сумма для приобретения вредоносных бинарных файлов. Затем рынки, работающие по схеме «оплата за каждый устанавливаемый экземпляр» (pay-per-install – PPI), предоставляют дешевые и надежные каналы распространения вредоносного ПО. Даже без организованных платформ дистрибуции вредоносное ПО может постоянно распространяться в широких масштабах через веб-среду, электронную почту, а также с помощью методов социальной инженерии. После распространения вредоносного ПО в группе ничего не подозревающих жертв злоумышленники могут извлечь потенциально гигантскую выгоду благодаря чрезвычайно высокому спросу на нелегальном рынке на похищенные удостоверения личности, номера кредитных карт и прочие секретные данные, а также получить доход от незаконно распространяемой рекламы.

Авторы вредоносных программ – это в большинстве своем опытные и талантливые разработчики, которые работают как для себя, так и для организованных нелегальных групп. Но большинство распространителей не является авторами программ. Распространители вредоносного ПО, как правило, приобретают нужные им программы на нелегальных онлайновых рынках и форумах. Некоторые более компетентные в техническом отношении дистрибьюторы похищают вредоносное ПО у других авторов и адаптируют его для личных целей. Семейство

экземпляров вредоносного ПО может демонстрировать одинаковую функциональность и происходить от одного общего «ствола генеалогического дерева», развивающегося со временем, но не все изменения в различные версии программ могут быть внесены одним и тем же автором (или группой). Новые вариации в семействе вредоносного ПО могут разрабатываться независимо друг от друга, при этом авторы исходной версии остаются неизвестными. Доступ к исходному коду или возможность выполнения реверс-инженерии и реассемблирования программ позволяют без труда вносить изменения всем желающим и распространять модифицированные программы как новое вредоносное ПО.

По сравнению с потенциальными выгодами, расходы по приобретению и распространению вредоносного ПО чрезвычайно малы. В качестве примера рассмотрим ПО для шантажа. Программы для шантажа предоставляют в распоряжение злоумышленника простой процесс получения денег. Специализированное ПО для шантажа (позволяющее покупателю вставлять сообственные угрожающие сообщения и адреса кошельков для биткойнов перед началом операции) можно приобрести на подпольных рынках за несколько десятков долларов. Затраты составляют приблизительно 180 долл. на тысячу успешных установок шантажистского ПО на компьютеры в атакуемом районе. Если требование выкупа в сумме 50 долл. отправить на каждый зараженный компьютер и 10 % жертв (по самой скромной оценке) предпочтут заплатить требуемую сумму, то предполагаемая прибыль шантажиста более чем в 25 раз превысит начальные расходы. Такая чрезвычайно выгодная «бизнес-модель» вполне объясняет широкое распространение компьютерного шантажа в последние несколько лет.

Важно отметить, что большинство незаконных операций позволяет получить лишь неточную, весьма искаженную картину экономической ситуации, поскольку подобные операции не контролируются действующими законами и юридическими нормами. Торговцы наркотиками, зарабатывающие на этом пагубном пристрастии, могут использовать весьма жесткую схему поставок и распространения, обеспечивающую астрономический рост их доходов. Грабители, вымогающие деньги у жерт под угрозой насилия, могут быть уверены в прибыльности своих действий. Различие между этими примерами и экономикой вредоносного ПО – это различие в определении субъекта при классификации преступления и статьи действующего закона. Почти невозможно с полной уверенностью определить ответственность конкретного лица за кибератаку или производство вредоносного ПО, следовательно, практически невозможно наказать преступника в рамках законности. Эта особенность делает распространение вредоносного ПО одним из самых прибыльных и наименее рискованных видов незаконной деятельности, которые когда-либо существовали.

Современные процессы выполнения кода

Теперь рассмотрим подробнее, как создаются и выполняются основные типы современных программ, а также возможности исследования бинарного кода и процесса выполнения программ, чтобы научиться понимать их внутреннее устройство и принципы работы без доступа к исходному коду.

Ниже приводится описание процесса выполнения кода большого класса компьютерных программ общего назначения, характерного для многих современных языков программирования и платформ, обеспечивающих выполнение кода. Это никоим образом нельзя

считать исчерпывающим или всеобъемлющим описанием того, как выполняются все типы программ. Огромная и разнообразная экосистема программных сред и подсистем времени выполнения подразумевает многочисленные тонкие и малозаметные различия при выполнении кода в разных средах. Но многие рассматриваемые здесь концепции можно обобщить и провести аналогии с другими типами процессов выполнения кода.

В целом существуют два типа выполнения кода: выполнение компилированного кода (compiled execution) и выполнение интерпретируемого кода (interpreted execution). При выполнении компилируемого кода предварительно написанный исходный код переводится в требуемые машинные инструкции при помощи последовательности преобразований[1] (часто обозначаемой термином «процесс сборки ПО» (software build process)). Полученные машинные инструкции помещаются в бинарные файлы, которые в дальнейшем могут выполняться непосредственно аппаратурой компьютера[2]. При выполнении интепретируемого кода предварительно написанный исходный код (иногда обозначаемый термином «скрипт» (script)) переводится в некоторый промежуточный формат, в дальнейшем передаваемый в интерпретатор (interpreter) для выполнения программы. Интепретатор отвечает за выполнение инструкций программы на аппаратуре, для которой написана программа. Для каждой реализации предусмотрен собственный промежуточный формат кода, но в большинстве случаев это один из вариантов байт-кода (bytecode) (бинарные машинные инструкции), которые будут выполняться на виртуальной машине (virtual machine)[3]. Некоторые реализации представляют собой гибридный механизм компилируемого и интерпретируемого выполнения с частым использованием динамической или JIT-компиляции (just-in-time (JIT) compilation), когда байт-код интерпретатора компилируется в машинные инструкции в реальном времени (т. е. непосредственно во время выполнения)[4].

На рис. 4.1 показана общая схема процесса выполнения кода для некоторых современных реализаций ПО.

Рассмотрим более подробно элементы блок-схемы на рис. 4.1:

- прямоугольники обозначают программу в ее различных состояниях;
- эллипсы обозначают этапы преобразования кода, которые переводят программу из одного состояния в другое;

---

[1] Примеры компилируемых языков: C/C++, Go, Haskell (и многие другие).

[2] Это не совсем точно: во многих случаях для выполнения бинарных файлов требуется еще и соответствующая среда выполнения (операционная система). – *Прим. перев.*

[3] Примеры языков с интепретатором байт-кода: Python, Ruby, Smalltalk, Lua.

[4] Java представляет собой интересный пример широко распространенного языка программирования, который может считаться и компилируемым, и интерпретируемым языком в зависимости от конкретной реализации. Java использует двухэтапный процесс компиляции: сначала написанный человеком исходный код Java компилируется в байт-код соответствующим компилятором, затем полученный байт-код выполняется виртуальной машиной Java virtual machine (JVM). Большинство современных виртуальных машин JVM использует JIT-компиляцию для трансляции байт-кода в аппаратные машинные инструкции, которые выполняются непосредственно на компьютере. В некоторых других реализациях JVM байт-код может интерпретироваться непосредственно виртуальной машиной, подобно тому как работают обычные интерпретируемые языки.

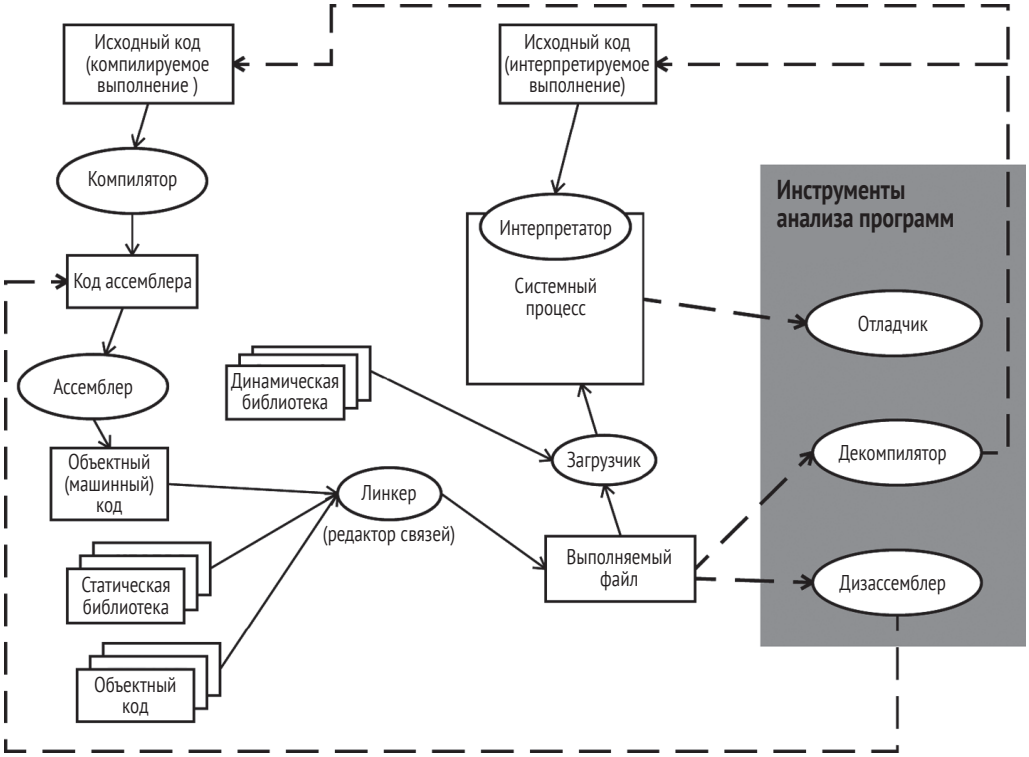


Рис. 4.1 ❖ Блок-схема выполнения кода и анализа программ

- сплошные стрелки между элементами блок-схемы обозначают преобразование кода из исходного (написанного человеком) в форму, которая может быть выполнена непосредственно аппаратурой компьютера (или операционной системой);
- серый прямоугольник содержит некоторые инструментальные средства, которые могут использоваться в реверс-инженерии для исследования статического или динамического состояния бинарного кода или работающей программы (это показано штриховыми стрелками) и создавать удобные условия наблюдения за процессом выполнения кода.

ВЫПОЛНЕНИЕ КОМПИЛИРУЕМОГО КОДА В качестве примера рассмотрим фрагмент кода на языке C, выполняющего простую арифметическую операцию, и пройдем по этапам процесса сборки компилируемой версии программы. В соответствии с блок-схемой на рис. 4.1 проследим путь программы от начального состояния «Исходный код (компилируемое выполнение)». Ниже приведен исходный код, для которого необходимо провести процесс сборки. Код сохранен в файле *add.c*[1]:

[1] Код этого примера можно найти в подкаталоге *chapter4/code-exec-eg/c* (**https://github.com/oreilly-mlsec/book-resources/tree/master/chapter4/code-exec-eg/c**) в репозитории кода для данной книги.

```
#include <stdio.h>

int main() {
    // Прибавить 1 к значению переменной x
    int x = 3;
    printf("x + 1 =  %d", x + 1);
    return 0;
}
```

1. Первый этап процесса сборки невелик, но важен: препроцессинг (preprocessing) (на рис. 4.1 не показан). В языке C строки, начинающиеся с символа #, интерпретируются препроцессором как его собственные директивы (**https://gcc.gnu.org/onlinedocs/cpp/**). Препроцессор просто выполняет проход по исходному коду и воспринимает эти директивы как макрокоманды (macros), подготавливая код для компиляции посредством вставки содержимого указанных включаемых (included) файлов и удаляя комментарии, а также выполняя некоторые другие подготовительные действия. Для просмотра результатов этапа препроцессинга можно выполнить следующую команду:

   ```
   > cc -E add.c

   [большая группа строк пропущена для краткости]
   extern void funlockfile (FILE *__stream)
                            __attribute__ ((__nothrow__ , __leaf__));
   # 942 "/usr/include/stdio.h" 3 4

   # 2 "add.c" 2

   # 3 "add.c"
   int main() {

       int x = 3;
       printf("x + 1 =  %d", x + 1);
       return 0;
   }
   ```

   Отметим, что вывод результата препроцессинга содержит множество строк кода, которых нет в исходном файле *add.c*. Препроцессор заменил строку `#include <stdio.h>` на соответствующее содержимое файла из стандартной библиотеки языка C `stdio.h`. Также отметим, что строка комментария из исходного кода отсутствует в выведенном результате.
2. Следующий этап процесса сборки – компиляция (compilation). Здесь компилятор выполняет преобразование кода, обработанного препроцессором, в код ассемблера. Сгенерированный код ассемблера предназначен для целевой архитектуры процессора, поэтому содержит инструкции, которые процессор (CPU) способен понять и выполнить. Инструкции ассемблера обязательно должны принадлежать к набору инструкций, распознаваемых целевым процессором. Для просмотра результата работы компилятора языка C с сохранением сгенерированного кода ассемблера в файле можно выполнить следующую команду:

   ```
   > cc -S add.c
   ```

Ниже показан код ассемблера, сгенерированный конкретной версией C-компилятора из комплекта GCC (GNU Compiler Collection)[1], предназначенной для 64-битовой системы Linux (x86_64-linux-gnu):

```
> cat add.s

    .file   "add.c"
    .section    .rodata
.LC0:
    .string "x + 1 =  %d"
    .text
    .globl  main
    .type   main, @function
main:
.LFB0:
    .cfi_startproc
    pushq    %rbp
    .cfi_def_cfa_offset 16
    .cfi_offset 6, -16
    movq     %rsp,  %rbp
    .cfi_def_cfa_register 6
    subq    $16,  %rsp
    movl    $3, -4( %rbp)
    movl    -4( %rbp),  %eax
    addl    $1,  %eax
    movl     %eax,  %esi
    movl    $.LC0,  %edi
    movl    $0,  %eax
    call    printf
    movl    $0,  %eax
    leave
    .cfi_def_cfa 7, 8
    ret
    .cfi_endproc
.LFE0:
    .size   main, .-main
    .ident  "GCC: (Ubuntu 5.4.0-6ubuntu1~16.04.4) 5.4.0 20160609"
    .section    .note.GNU-stack,"",@progbits
```

Приведенный выше результат компиляции может выглядеть совершенно непонятным, если вы не разбираетесь в ассемблерном коде (в данном случае – код ассемблера x64). Но даже с минимальным знанием ассемблера появляется возможность получения полного представления о том, что делает эта программа, на основе только этого ассемблерного кода. Обратите внимание на две строки, выделенные полужирным шрифтом в приведенном примере: `addl …` и `call printf` – достаточно легко понять, что программа выполняет сложение, затем вызывает функцию вывода результата. Большинство других строк выполняет вспомогательную работу – размещение и извлечение значений в регистрах процессора и в ячейках памяти, где к ним

[1] В рассматриваемом примере использован компилятор GCC version 5.4.0 201606609 (Ubuntu 5.4.0-6ubuntu1~16.04.4 (**https://gcc.gnu.org/onlinedocs/5.4.0/**).

могут получить доступ другие функции. В любом случае анализ кода ассемблера – это обширная тема для обсуждения, но здесь мы не будем углубляться в подробности[1].

3. После генерации кода ассемблера начинают работу компонент assembler для преобразования инструкций ассемблера в объектный код (машинный код). Результатом работы компонента ассемблирования является набор машинных инструкций, которые будет непосредственно выполнять целевой процессор:

```
> cc -c add.c
```

Эта команда создает объектный файл *add.o*. Содержимым этого файла является бинарный формат, который трудно расшифровать, но мы все же попробуем его просмотреть. Это можно сделать с помощью инструментальных утилит, таких как `hexdump` или `od`. Утилита `hexdump` по умолчанию выводит содержимое заданного файла в шестнадцатеричном формате. В первом столбце вывода указано смещение внутри файла от его начала (также в шестнадцатеричном формате), что помогает найти соответствующее содержимое:

```
> hexdump add.o

0000000 457f 464c 0102 0001 0000 0000 0000 0000
0000010 0001 003e 0001 0000 0000 0000 0000 0000
0000020 0000 0000 0000 0000 02b8 0000 0000 0000
0000030 0000 0000 0040 0000 0000 0040 000d 000a
0000040 4855 e589 8348 10ec 45c7 03fc 0000 8b00
0000050 fc45 c083 8901 bfc6 0000 0000 00b8 0000
0000060 e800 0000 0000 00b8 0000 c900 78c3 2b20
    [часть вывода пропущена для краткости]
00005c0 0000 0000 0000 0000 0000 0000 0000 0000
00005d0 01f0 0000 0000 0000 0013 0000 0000 0000
00005e0 0000 0000 0000 0000 0001 0000 0000 0000
*
00005f8
```

Утилита `od` (сокращение от octal dump – вывод в восьмеричном формате) выводит содержимое файлов в восьмеричном и других форматах. Ее вывод может оказаться немного более удобным для чтения, особенно если вы не специалист по работе с шестнадцатеричными числами:

```
> od -c add.o

...0000 177   E   L   F 002 001 001  \0  \0  \0  \0  \0  \0  \0  \0
...0020 001  \0   >  \0 001  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
...0040  \0  \0  \0  \0  \0  \0  \0  \0 270 002  \0  \0  \0  \0  \0
...0060  \0  \0  \0  \0   @  \0  \0  \0  \0  \0   @  \0  \r  \0  \n  \0
...0100   U   H 211 345   H 203 354 020 307   E 374 003  \0  \0  \0 213
                [часть вывода пропущена для краткости]
```

[1] Для глубокого изучения ассемблера существует много хороших книг, например Assembly Language Step-by-Step: Programming with Linux, 3rd ed., by Jeff Duntemann (издательство Wiley) and The Art of Assembly Language, 2nd ed., by Randall Hyde (издательство No Starch Press).

```
...2700  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0  \0
...2720 360 001  \0  \0  \0  \0  \0  \0 023  \0  \0  \0  \0  \0  \0  \0
...2740  \0  \0  \0  \0  \0  \0  \0  \0 001  \0  \0  \0  \0  \0  \0  \0
...2760  \0  \0  \0  \0  \0  \0  \0  \0
...2770
```

Это позволяет в определенной степени разобраться непосредственно в структуре бинарного файла. Например, отметим, что почти в самом начале файла находятся символы E, L, F. Это означает, что ассемблер сгенерировал ELF-файл (Executable and Linkable Format, точнее ELF64). Каждый ELF-файл начинается с заголовка, содержащего некоторые характеристики файла, включая собственно тип этого файла. Существуют специализированные утилиты, например `readelf`, которые помогают выполнить синтаксический анализ всей информации, содержащейся в этом заголовке:

```
> readelf -h add.o

ELF Header:
  Magic:   7f 45 4c 46 02 01 01 00 00 00 00 00 00 00 00 00
  Class:                             ELF64
  Data:                              2's complement, little endian
  Version:                           1 (current)
  OS/ABI:                            UNIX - System V
  ABI Version:                       0
  Type:                              REL (Relocatable file)
  Machine:                           Advanced Micro Devices X86-64
  Version:                           0x1
  Entry point address:               0x0
  Start of program headers:          0 (bytes into file)
  Start of section headers:          696 (bytes into file)
  Flags:                             0x0
  Size of this header:               64 (bytes)
  Size of program headers:           0 (bytes)
  Number of program headers:         0
  Size of section headers:           64 (bytes)
  Number of section headers:         13
  Section header string table index: 10
```

4. На этом этапе попробуем выполнить объектный файл, сгенерированный ассемблером[1]:

```
> chmod u+x add.o
> ./add.o
bash: ./add.o: cannot execute binary file: Exec format error
```

Почему выводится сообщение об ошибке: `Exec format error`? В объектном коде, сгенерированном ассемблером, отсутствуют весьма важные части программы, в обязательном порядке требуемые для ее выполнения. Кро-

[1] Для выполнения бинарного файла в Unix-подобной системе необходимо обладать правами на выполнение этого файла. `chmod` – это команда и системный вызов, позволяющие изменять права доступа к файлам в Unix-системах. В частности, аргумент `+x` определяет, что необходимо добавить право на выполнение (execute) для указанного файла.

ме того, секции программы не скомпонованы надлежащим образом, поэтому библиотечные и программные функции не могут быть правильно вызваны. Компоновка или редактирование связей (linking) – завершающий этап процесса сборки программы, который устраняет все перечисленные выше проблемы. В нашем случае редактор связей или просто линкер (linker) вставляет объектный код для библиотечной функции `printf` в этот бинарный файл. Выполним еще один вызов компилятора `cc` для генерации полноценной окончательной версии выполняемого бинарного файла (здесь необходимо явно указать имя выходного файла `add`, иначе компилятор `cc` использует имя по умолчанию `a.out`), затем еще раз попытаемся выполнить программу:

```
> cc -o add add.c
> chmod u+x add
> ./add

x + 1 = 4
```

Этот этап завершает процесс сборки простой программы на языке C – от исходного кода до выполняемого бинарного файла.

В предыдущем примере файл `stdio.h` из внешней библиотеки был статически связан (linked), т. е. включен в бинарный файл[1]. Это означает, что при компиляции выполняется статическое связывание с остальным кодом библиотеки (пакета). Некоторые языки и реализации допускают динамическое (dynamic) связывание с внешними библиотеками, т. е. в скомпилированный выполняемый код включаются только ссылки на библиотечные компоненты, а не их бинарный код. Во время выполнения вызывается загрузчик (loader), просматривающий выполняемую программу в поисках ссылок на динамически связанные библиотеки (или совместно используемые библиотеки[2] с расширениями .so, .dll и т. п.), а затем реализующий эти ссылки, определяя место расположения требуемых библиотек в системе. Но здесь мы не будем более подробно рассматривать механизмы загрузки динамически связываемых библиотек.

ВЫПОЛНЕНИЕ ИНТЕРПРЕТИРУЕМОГО КОДА В качестве примера реализации интерпретируемого языка программирования рассмотрим типовой процесс выполнения скрипта на языке Python. Отметим, что существует несколько разных реализаций языка Python с различными процессами выполнения кода. В при-

---

[1] Авторы не совсем точны: `stdio.h` – это заголовочный файл стандартной библиотеки, содержащий необходимые объявления библиотечных функций. Код самих функций содержится в объектных файлах библиотеки, которые линкер связывает (в данном случае статически) с бинарным файлом программы. – *Прим. перев.*

[2] Чаще всего в Unix-подобных системах и их вариациях (таких как Linux и macOS) используется термин «shared libraries» или «shared objects» для обозначения динамически связываемых библиотек. В ОС Windows применяется термин «dynamically linked libraries» (DLL). В некоторых языковых средах (например, Lua) существует тонкое различие между совместно используемыми (shared) и динамически связываемыми (dynamic) библиотеками: совместно используемая библиотека или совместно используемый объектный файл представляет собой специализированный тип динамически связываемой библиотеки, единственная копия которой совместно используется несколькими активными процессами.

водимом ниже примере рассматривается CPython[1] – стандартная, оригинальная реализация Python, написанная на C. Мы воспользуемся версией Python 3.5.2 из дистрибутива Ubuntu 16.04. Снова обращаясь к блок-схеме на рис. 4.1, проследим полный путь программы, начиная с состояния «Исходный код (интерпретируемое выполнение)»[2]:

```
class AddOne():
    def __init__(self, start):
        self.val = start
    def res(self):
        return self.val + 1

def main():
    x = AddOne(3)
    print('3 + 1 = {} '.format(x.res()))

if __name__ == '__main__':
    main()
```

1. Начнем с того, что этот исходный код на языке Python сохранен в файле *add.py*. Запуск скрипта с передачей имени его файла как аргумента в интерпретатор Python дает ожидаемый результат:

   ```
   > python add.py
   3 + 1 = 4
   ```

   Разумеется, это излишне усложненный способ сложения двух чисел, но такой пример дает нам возможность исследовать внутренний механизм выполнения программ в среде Python. Написанный человеком исходный код Python компилируется в промежуточный формат, называемый байт-кодом (bytecode), независимым от платформы внутренним представлением программы. Можно видеть скомпилированные модули Python (файлы с расширением *.pyc*[3]), создаваемые, если скрипт импортирует внешние модули и может быть записан в целевой каталог[4]. В нашем примере внешние модули не импортируются, поэтому файлы *.pyc* не создаются. Чтобы проследить этот этап сборки программы, можно принудительно определить создание такого файла, используя для этого модуль `py_compile`:

   ```
   > python -m py_compile add.py
   ```

---

[1] Не следует путать эту реализацию с Cython – расширением языка Python, написанным на C и обладающим функциональными возможностями, позволяющими устанавливать связи с внешними библиотеками языка C.

[2] Код этого примера можно найти в подкаталоге *chapter4/code-exec-eg/python* (**https://github.com/oreilly-mlsec/book-resources/tree/master/chapter4/code-exec-eg/python**) в репозитории кода для данной книги.

[3] Если код Python компилируется с включенной оптимизацией, то создаются файлы с расширением *.pyo*. По существу, файл *.pyo* – это то же самое, что файл *.pyc*.

[4] Скомпилированные файлы создаются как оптимизированные для сокращения времени запуска программы. В версиях Python, более ранних, чем 3.2, автоматически генерируемые файлы *.pyc* создавались в том же каталоге, что и основной файл *.py*. В последующих версиях эти файлы создавались в подкаталоге *pycache*, и им присваивались другие имена в зависимости от интерпретатора Python, который их создал.

Эта команда создает файл *.pyc*, в котором содержится скомпилированный байт-код нашей программы. В версии Python 3.5.2 скомпилированный файл создается и сохраняется в файле с именем *pycache/add.cpython-35.pyc*. Далее можно просматривать содержимое этого бинарного файла, удалив заголовок и выполнив демаршалинг файла в структуру types.CodeType (**https://docs.python.org/3.2/library/types.html?highlight=types.codetype#types.CodeType**):

```
import marshal
import types

# Преобразование big-endian 32-битового массива байтов в тип long
def to_long(s):
    return s[0] + (s[1] << 8) + (s[2] << 16) + (s[3] << 24)

# Вывод иерархии имен из кода и номеров строк
def inspect_code(code, indent='    '):
    print('{}{}(line:{})'.format(indent,
        code.co_name, code.co_firstlineno))
    for c in code.co_consts:
        if isinstance(c, types.CodeType):
            inspect_code(c, indent + '    ')

f = open('__pycache__/add.cpython-35.pyc', 'rb')

# Чтение заголовка файла .pyc
magic = f.read(4)
print('magic: {}'.format(magic.hex()))
mod_time = to_long(f.read(4))
print('mod_time: {}'.format(mod_time))

# Только для Python версий >=3.3 файлы .pyc содержат заголовок source_size ❶
source_size = to_long(f.read(4))
print('source_size: {}'.format(source_size))

print('\ncode:')
code = marshal.load(f)
inspect_code(code)

f.close()
```

❶ В файлах *.pyc* для Python версии 3.2 и более ранних файлах имеется заголовок, содержащий два 32-битовых big-endian (с порядком битов от старшего к младшему) числа (**https://www.python.org/dev/peps/pep-3147/**), за которыми следует маршализованный объект кода. В версиях 3.3 и более поздних в заголовок также включено новое 32-битовое поле, в котором кодируется размер файла исходного кода (это увеличивает размер заголовка с 8 до 12 байтов в Python 3.3 по сравнению с более ранними версиями).

Выполнение приведенного выше скрипта дает следующий результат:

```
magic: 160d0d0a
mod_time: 1493965574
source_size: 231

code:
    <module>(line:1)
```

```
AddOne(line:1)
    __init__(line:2)
    res(line:4)
main(line:7)
```

В объекте CodeType содержится гораздо больше информации, которая здесь не выводится, но даже то, что показано в этом выводе, дает представление о структуре бинарного байт-кода в общих чертах.

2. Сгенерированный байт-код выполняется механизмом времени выполнения виртуальной машины Python. Отметим, что этот байт-код не является бинарным машинным кодом. Это специализированные для программной среды Python коды операций (opcodes), интерпретирующиеся виртуальной машиной Python, которая затем транслирует эти коды в машинные инструкции. С помощью функции `dis.disassemble()` (**https://docs.python.org/3.5/library/dis.html**) можно дизассемблировать объект `code`, созданный выше, и получить следующий результат:

```
> import dis
> dis.disassemble(code)
```

```
  1           0 LOAD_BUILD_CLASS
              1 LOAD_CONST               0 (< code object AddOne at
                                           0x7f78741f7930, file
                                           "add.py", line 1> )
              4 LOAD_CONST               1 ('AddOne')
              7 MAKE_FUNCTION            0
             10 LOAD_CONST               1 ('AddOne')
             13 CALL_FUNCTION            2 (2 positional,
                                           0 keyword pair)
             16 STORE_NAME               0 (AddOne)

  7          19 LOAD_CONST               2 (< code object main at
                                           0x7f78741f79c0, file
                                           "add.py", line 7> )
             22 LOAD_CONST               3 ('main')
             25 MAKE_FUNCTION            0
             28 STORE_NAME               1 (main)

 11          31 LOAD_NAME                2 (__name__)
             34 LOAD_CONST               4 ('__main__')
             37 COMPARE_OP               2 (==)
             40 POP_JUMP_IF_FALSE       50

 12          43 LOAD_NAME                1 (main)
             46 CALL_FUNCTION            0 (0 positional,
                                           0 keyword pair)
             49 POP_TOP
        > >  50 LOAD_CONST               5 (None)
             53 RETURN_VALUE
```

Кроме того, можно получить результаты, показанные на двух предыдущих этапах, если вызвать модуль `trace` (**https://docs.python.org/2/library/trace.html**) непосредственно из командной строки: `python -m trace add.py`.

Сразу можно заметить сходство между приведенным выше результатом дизассемблирования и кодом ассемблера x86, который рассматривался в предыдущем подразделе. Виртуальная машина Python считывает этот байт-код и переводит его в машинный код[1], который выполняется на целевой архитектуре. Таким образом, процесс выполнения кода завершен.

Процесс выполнения кода для интерпретируемых языков короче, потому что не требует этапа сборки или компиляции: код можно выполнять сразу после его написания. Байт-код Python невозможно выполнить напрямую на целевой аппаратуре, сначала требуется его интерпретация виртуальной машиной Python и последующая трансляция в машинный код. Для этого процесса характерна некоторая потеря эффективности и производительности по сравнению с языками более низкого уровня, такими как C. Тем не менее следует отметить, что в процессе выполнения исходного кода из файла Python все же присутствует в определенной степени компиляция, так что виртуальной машине Python не нужно выполнять повторный анализ и синтаксический разбор каждой команды исходного кода многократно во время выполнения программы. Запуск программной среды Python в режиме интерактивной командной оболочки больше похож на модель реализации полностью интерпретируемого языка, поскольку каждая строка анализируется и снитаксически разбирается во время выполнения.

При работе с исходным кодом, написанным человеком, можно с легкостью идентифицировать и проанализировать особые свойства и цели компонента ПО, которые позволяют точно классифицировать его по семейству и функциональности. Но поскольку доступ к исходному коду предоставляется редко, необходимо применять другие средства извлечения информации о конкретной программе. Хорошо понимая современные процессы выполнения кода, можно перейти к рассмотрению различных способов статического и динамического (непосредственно во время выполнения) анализа вредоносного ПО. Код проходит четко определенный путь от создания его автором до выполнения. Внимательное изучение любого пункта этого пути может дать весьма полезную информацию о программе.

ТИПОВОЙ ПРОЦЕСС АТАКИ ВРЕДОНОСНОГО ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

При исследовании и классификации вредоносного ПО важно понимать, что именно делает вредоносная программа и как возникла уязвимость. Как уже отмечалось в главе 1, разные типы вредоносного ПО используют различные методы распространения, преследуют разнообразные цели и создают разные уровни угрозы для частных лиц и организаций. Тем не менее возможно дать определенную характеристику типового процесса (потока) атаки вредоносного ПО, как показано на рис. 4.2.

В фазе 1 начальные разведывательные действия обычно пассивны и используют косвенные методы определения свойств и характеристик цели. После этого начинается активная рекогносцировка, например сканирование портов и сбор более точной актуальной информации о цели, т. е. поиск слабых мест для про-

[1] В интерпретаторе CPython преобразование кодов операций Python в машинные инструкции выполняется достаточно просто. Специальная команда преобразования битов (bit switch statement) отображает каждую строку кода операции Python в код на языке C, который затем выполняется на целевой машине, после того как ассемблер транслирует его в машинный код.

никновения. Слабым местом может быть открытый порт, с которым работает немодифицированное уязвимое ПО, или сотрудник, который может стать объектом фишинговых атак. Результатом использования слабых мест может стать успешное проникновение вредоносной программы сквозь защитные барьеры. После успешного проникновения цель становится жертвой.

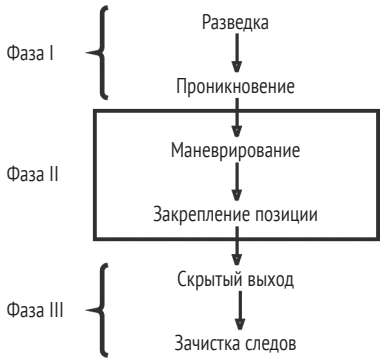


Рис. 4.2 ❖ Типовой процесс (поток) атаки вредоносного ПО

В фазе 2 вредоносная программа уже находится в среде жертвы. С помощью процесса внутренней рекогносцировки и создания опорного пункта на хосте (это называется горизонтальным перемещением (horizontal movement)) вредоносная программа может начать маневрирование по сети для поиска важных хостов. Затем она закрепляется в программной среде, используя такие средства, как создание скрытых «черных ходов» для будущих сеансов доступа, или позиционируя себя как постоянный фоновый процесс-демон.

В фазе 3 вредоносная программа готовится удалить себя из среды жертвы и не оставить никаких следов. Для вредоносного ПО, которое занимается похищением секретной информации любого типа, на этапе скрытого выхода похищенные данные (например, идентификационные данные пользователя, номера кредитных карт, секретная бизнес-логика) передаются на удаленный сервер. Наконец, когда задача выполнена окончательно, вредоносная программа может принять решение о полном удалении самой себя и зачистке всех следов своей деятельности на компьютере жертвы.

В зависимости от типа вредоносного ПО некоторые или все этапы во всех трех фазах могут быть особенно важными. Кроме того, вредоносное ПО часто демонстрирует определенные типы поведения, которые необходимо хорошо знать и понимать их сущность. Рассмотрим подробнее эти типы поведения:

- маскировка своего присутствия: вредоносное ПО часто использует методики упаковки (сжатия) и шифрования для придания своему коду более компактной и замаскированной формы. Цель – избежать обнаружения и препятствовать успешному анализу экспертов по защите;
- стремление к выполнению своей функции: для эффективного выполнения своей функции вредоносная программа должна обеспечить достаточную степень живучести, чтобы не быть уничтоженной из-за изменений в системе или обнаруженной человеком-администратором. Широко используются

методики защитных уловок, такие как скрытая побочная загрузка DLL (DLL sideloading – **https://attack.mitre.org/techniques/T1073/**) и принудительное завершение антивирусных процессов. Некоторым типам вредоносного ПО необходима возможность маневрирования по сети в «горизональном направлении». Большинство типов вредоносных программ пытается в той или иной форме расширить свои привилегии (используя для этого уязвимости ПО/ОС, такие как переполнение буферов памяти, или приемы социальной инженерии, применяемые к конечным пользователям) для получения прав администратора на атакуемой платформе;

- сбор данных и «звонок-оповещение»: после того как вредоносная программа завершила сбор всех необходимых данных (секретные регистрационные данные сервера/приложения, журналы сеансов доступа в веб, записи баз данных и т. п.), она передает эти данные на внешний пункт сбора. Также может быть выполнен «звонок-оповещение» (phone home) на удаленный сервер управления и контроля (C&C – command-and-control) с целью получения дальнейших инструкций.

Генерация признаков

Специалист по исследованию данных гораздо больше времени тратит на сбор данных и размещение их в одном месте и на форматирование собранных данных, чтобы получить возможность более эффективного их использования, по сравнению с затратами времени на создание классификаторов или выполнение статистического анализа. Далее в этой главе будет рассматриваться методика извлечения и конструирования признаков с использованием вредоносных программ и выполняемых бинарных файлов в качестве примера (образца). Начнем с обзора трудностей при сборе данных в форме, подходящей для извлечения признаков. Затем будет подробно рассматриваться задача генерации признаков для классификации вредоносного ПО с помощью богатого набора методик анализа выполняемых файлов, среди которых есть вполне подходящие для автоматизации.

Конструирование признаков чрезвычайно важно для всех приложений машинного обучения, но почему мы обращаем особое внимание на конструирование признаков бинарных данных? Бинарные данные представляют собой своего рода «общий знаменатель» самого низкого уровня представления данных. Все другие формы информации могут быть представлены в бинарном формате, а извлечение данных из бинарных файлов – это задача интерпретации битов, из которых состоят бинарные файлы. Извлечение и конструирование признаков – это процесс интерпретации необработанных исходных данных для генерации тех их характеристик, которые наилучшим образом представляют природу (сущность) распределения. Не существует более сложного для анализа и более подходящего для обеспечения защиты формата данных, чем выполняемые бинарные файлы.

Важность процесса сбора данных и конструирования признаков в машинном обучении невозможно переоценить. Специалисты по исследованию данных и инженеры машинного обучения иногда обнаруживают, что находятся в ситуации, в которой у них имеется минимальная возможность или вовсе нет возможности как-либо повлиять на методику и процесс сбора данных. Это чрезвычайно неприятное положение, поскольку самые крупные прорывы и усовершенствования

технологии машинного обучения и науки о данных чаще всего основаны на улучшении качества исходных «сырых» данных, а не на использовании затейливых алгоритмов или проектировании более эффективных систем. Какой бы ни была задача, самое важное при ее решении – серьезно заняться трудоемкой и скучной обработкой источников исходных «сырых» данных для поиска наилучшего способа извлечения информации, необходимой для получения хороших результатов. Если алгоритм машинного обучения работает неудовлетворительно, всегда следует помнить о том, что причиной этого может быть плохое качество данных, а не недостатки самого алгоритма.

В любом случае сбор данных часто может стать самой трудоемкой, требующей немалых затрат времени и средств областью применения науки о данных. Важно предварительно продумать и спроектировать гибкую и эффективную архитектуру для сбора данных, потому что она способна существенно ускорить процесс создания системы машинного обучения. Значительные преимущества можно получить, выполняя широкомасштабные исследования с целью поиска наилучшего способа сбора данных и определения объектов данных, которые необходимо собирать и которые нужно отсеивать. Рассмотрим некоторые важные вопросы, касающиеся сбора данных для машинного обучения.

Сбор данных

Если просто позволить приложению принимать бесконечный поток информации из интернета, то вряд ли удастся достичь приемлемого качества данных для машинного обучения. В итоге будет собрана масса ненужных данных вместе с теми данными, которые действительно необходимы, но и они могут быть искажены или неточны. Ниже описаны некоторые положения, которые специалисты по исследованию данных используют для улучшения процесса сбора данных:

- важность знаний проблемной области: сбор данных для анализа вредоносного ПО на основе машинного обучения непременно требует весьма специфического набора знаний проблемной (предметной) области, отличающегося от других прикладных направлений, таких как компьютерное зрение. Даже в новой области знаний (т. е. при недостатке знаний в предметной области) иногда полезно осмыслить задачу с другой точки зрения. Глубокое экспертное исследование прикладной предметной области может помочь очень быстро определить важные признаки, которые нужно собрать. Таким образом, алгоритм обучения получит действительно важные элементы данных.

  В сфере обеспечения безопасности полезны глубокое знание работы компьютерных сетей, основ функционирования операционных систем, процессов выполнения программного кода и т. п., если машинное обучение применяется в этих сферах. Иногда трудно достичь уровня эксперта в разнообразных предметных областях знаний и получить практический опыт решения специфических задач за чрезвычайно короткое время. В таких ситуациях могут оказаться весьма полезными консультации экспертов в соответствующих предметных областях, проводимые перед проектированием схем сбора данных и конструирования признаков;
- масштабируемые процессы сбора данных: для получения действительно полезных результатов часто необходимо предоставить выбранным алго-

ритмам машинного обучения огромные объемы данных. Возможно, проще вручную извлечь признаки из десятка экземпляров данных, но при наличии миллиона и более элементов выборки задача существенно усложняется. Да и реверс-инженерия бинарных файлов представляет собой задачу, требующую колоссальных затрат времени и ресурсов. Ручная реверс-инженерия набора данных, состоящего из сотен тысяч разнообразных образцов вредоносного ПО, влечет за собой непомерные и недопустимые издержки. Следовательно, чрезвычайно важно тщательно продумать процедуры автоматизации процессов сбора данных, прежде чем начать увеличивать масштабы выполняемой операции. Но при сочетании знаний проблемной области и исследования данных вы всегда можете разработать методики, позволяющие сосредоточить усилия на автоматизации сбора только самых важных признаков, требуемых для конкретной задачи;

- валидация и неточность: как узнать, что собранные данные полны и корректны? Валидация (проверка) данных имеет первостепенную важность, поскольку систематические, постоянно повторяющиеся ошибки при сборе данных могут полностью исказить результаты последующего анализа и привести к катастрофическому состоянию всей системы машинного обучения. Но тщательно проверить входные данные алгоритмическим путем не так-то просто. Наилучшим способом выявления подобных проблем на раннем этапе является частое выполнение проверки случайной выборки собираемых данных вручную. Если результаты проверки не соответствуют вашим предположениям, то необходимо как можно быстрее найти главную причину и определить, является ли ошибка при сборе данных причиной этого несоответствия.

  Если собираемым данным присуща внутренняя неточность (необъективность), то в этом случае требуется большее внимание, поскольку ошибки труднее определить даже при исследовании вручную. Единственный способ надежного выявления подобной проблемы – недвусмысленно рассматривать ее как потенциальную причину плохих результатов машинного обучения. Например, если система классификации изображений животных в целом демонстрирует неплохую точность, но постоянно выдает ошибочные результаты для категорий птиц, то возможная причина этого состоит в том, что признаки, выбранные из исходных данных, в большей степени подходят для других животных, нежели для птиц, или в том, что собранные данные состоят только из изображений птиц в спокойном состоянии, а не в полете.

  В наборах данных о вредоносном ПО часто возникает проблема их неактуальности по причине слишком быстрых изменений природы элементов выборки со временем. Например, выборка по некоторому семейству вредоносного ПО, собранная в январе, может оказаться чрезвычайно нерепрезентативной по сравнению с выборкой, собранной в марте, если разработчики вредоносных программ этого семейства достаточно хитроумны, чтобы постоянно вносить изменения с целью избежать обнаружения по сигнатуре. В наборах данных о характеристиках систем защиты также часто возникает дисбаланс классов, так как иногда трудно подобрать приблизительно одинаковое количество обычных и вредоносных экземпляров выборки;

- итеративные эксперименты: машинное обучение представляет собой процесс итеративного экспериментирования, и этап сбора данных не является исключением. Если в какой-либо точке процесса вас остановили неудовлетворительные результаты, примените научный способ мышления и интепретируйте ситуацию как неудачный вариант управляемого эксперимента. Подобно тому, как ученый изменяет условия и параметры и повторяет эксперимент, вы должны сделать правильно обоснованное предположение о вероятной причине ошибки и попытаться повторить свой эксперимент.

Генерация признаков

Цель этой главы – представить разработку общей стратегии извлечения информации из сложных бинарных файлов различных форматов. Предлагаемая здесь стратегия обосновывается в подробном обсуждении процедуры отделения полного, подробно описанного набора признаков для одного конкретного типа бинарных файлов. В качестве практического примера мы выбрали бинарные файлы ОС Android из-за их возрастающей значимости в современном мире мобильной связи, а также потому, что методы анализа приложений Android можно без затруднений обобщить для анализа других форматов бинарных файлов, например для приложений для десктопа или мобильных устройств, выполняемых макросов для документов или подключаемых модулей браузера. Даже несмотря на то что некоторые из рассматриваемых здесь инструментальных средств и методик анализа являются специализированными для экосистемы Android, в большинстве случаев можно найти достаточно близкие их аналоги в других операционных средах.

При извлечении признаков для любой задачи машинного обучения необходимо всегда помнить о конечной цели. Некоторые задачи в гораздо большей степени полагаются на определенные признаки, нежели на другие, но здесь мы не будем обсуждать сравнительную важность и значимость признаков, поскольку подобные оценки неразрывно связаны с тем, как используются сгенерированные данные для достижения конкретной цели. Поэтому мы не рассматриваем извлечение признаков с точки зрения одной узкой задачи машинного обучения (классификация по семействам вредоносного ПО, классификация поведения, определение характеристик вредоносности и т. п.). Вместо этого мы рассмотрим генерацию признаков с более общих позиций, и нашей главной целью является генерация всех возможных описательных признаков из бинарного файла со сложным содержимым.

Анализ вредоносных программ в среде ОС Android

Операционная система Android широко распространена. На рынке операционных систем для смартфонов это самый сильный игрок[1]. Но именно из-за своей широкой распространенности Android представляет собой весьма привлекательную платформу для атак злоумышленников, пытающихся найти мак-

[1] В финансовых отчетах за 2016 г. было отмечено, что 81,7 % продаж смартфонов по всему миру пришлось на долю устройств с ОС Android (**https://www.gartner.com/en/newsroom/press-releases/2017-02-15-gartner-says-worldwide-sales-of-smartphones-grew-7-percent-in-the-fourth-quarter-of-2016**).

симальную область воздействия на потенциальные жертвы. В совокупности со свободными и открытыми для всех рынками приложений (по сравнению с замкнутой и ограниченной экосистемой приложений для Apple iOS) это означает, что Android быстро стал мобильной платформой, которую предпочитают авторы вредоносного ПО[1].

Изучая внутреннюю структуру и принципы работы приложений Android, можно применять методики реверс-инженерии для поиска признаков, которые могут помочь в идентификации и классификации вредоносного ПО. Подобные последовательные действия, выполняемые вручную, помогают сгенерировать значимые признаки для нескольких приложений Android, но такой подход плохо масштабируется, если необходимо применить ту же методику извлечения признаков к более крупным наборам данных. Поэтому при изучении приведенного здесь примера необходимо постоянно помнить о том, что простота автоматизации извлечения признаков не менее важна, чем изобилие и полнота выбранных признаков. Кроме определения признаков, которые должны быть извлечены, также весьма важно определить, как их извлечь наиболее эффективным и универсальным (в смысле масштабируемости) способом.

Обобщенная методология конструирования признаков должна быть проработана как можно тщательнее с учетом всех полезных представлений имеющихся данных. Если каждый элемент выборки состоит только из нескольких логических (Boolean) признаков, то не требуется сложная процедура извлечения признаков – достаточно использовать исходные необработанные данные как входные данные для алгоритмов классификации. Но если каждый экземпляр так же сложен и объемен, как программные приложения и выполняемые бинарные файлы, то предстоит большая работа. Относительно небольшой бинарный файл размером 1 Мб содержит 2^{23} битов информации, которые мгновенно превращаются в 8 388 608 различных возможных значений. Попытка выполнения задач классификации с использованием информации на уровне битов может быстро стать трудноразрешимой проблемой. Это не самое эффективное представление, поскольку данные содержат огромный объем избыточной информации, бесполезной для процесса машинного обучения. Поэтому необходимо применить некоторые знания предметной области, относящейся к структуре бинарных файлов (которую мы обсуждали в предыдущих разделах главы) и к способу выполнения этих файлов в системной среде, чтобы извлечь дескриптивные признаки более высокого уровня. Ниже мы подробно рассмотрим различные методы внутреннего исследования приложений ОС Android, не забывая о том, что многие из этих методов могут быть обобщены для решения задачи генерации признаков для других типов выполняемых бинарных файлов. В качестве общего набора инструментальных средств для анализа выполняемых бинарных файлов будут рассматриваться следующие методики:

---

[1] Это замечание вовсе не означает, что устройства с ОС Android по своей природе менее защищены, чем устройства под управлением iOS. В каждой операционной системе есть собственный набор документированных проблем по обеспечению безопасности. И для Android, и для iOS характерны явные принципиальные различия в открытости ПО и проверке приложений, поэтому нельзя с определенностью сказать, какой вариант лучше. В обеих операционных системах существует сравнимое количество уязвимостей, и стратегия защиты каждой экосистемы имеет свои достоинства и недостатки.

- статические методы:
  - анализ структуры;
  - статический анализ;
- динамический анализ:
  - анализ поведения;
  - отладка;
  - динамические контрольные измерения.

Приступим к практическому применению этих методик (не обязательно в перечисленном выше порядке) для анализа реально существующего вредоносного приложения ОС Android тем же способом, которым воспользовался бы опытный аналитик вредоносного ПО. Эта операция, выполняемая вручную, является первым и наиболее важным этапом процесса генерации признаков. В следующих разделах будет использоваться общее имя файла *infected.apk* для обозначения каждого анализируемого вредоносного пакета Android[1].

Java и механизм времени выполнения ОС Android

Несмотря на то что приложения ОС Android написаны на Java-подобном языке, существуют явные различия между Java API и Android API. В обычной среде выполнения Java исходный код компилируется в байт-код, который затем выполняется виртуальной машиной Java (JVM). В ранних версиях Android (до версии Android 4.4 KitKat) скомпилированный байт-код сохранялся в файлах с расширением .dex (Dalvik Executable)[2] и выполнялся виртуальной машиной Dalvik. Dalvik имеет регистровую архитектуру, тогда как архитектура JVM стековая. Так как виртуальная машина Dalvik предназначена для работы в средах с ограниченными ресурсами, таких как мобильные устройства и встроенные системы, она спроектирована так, чтобы использовать меньший объем памяти, и включает множество упрощений для большей эффективности. В новых версиях ОС Android механизм Android Runtime (ART) заменил Dalvik (**https://source.android.com/devices/tech/dalvik**) и стал новым стандартом среды выполнения программ Android. ART принимает на входе те же файлы с байт-кодом .dex, но выполняет больше операций оптимизации производительности (например, опережающую компиляцию (ahead-of-time compilation) во время установки) для повышения скорости выполнения приложений и более рационального использования ресурсов.

АНАЛИЗ СТРУКТУРЫ Приложения ОС Android упакованы в форме Android Package Kit (APK) файлов, которые в действительности являются архивными фай-

[1] Бинарный APK-файл Android вместе с декомпилированными файлами можно найти в каталоге chapter4/datasets в репозитории кода для данной книги (**https://github.com/oreilly-mlsec/book-resources/tree/master/chapter4/datasets**).

[2] Также используется расширение имени файла *.odex* для корректных выполняемых файлов Dalvik, отличающихся от обычных тем, что содержат оптимизированный байт-код Dalvik. После перехода от Dalvik к Android Runtime (ART) файлы *.odex* стали считаться устаревшими и в настоящее время не используются. Механизм ART применяет компиляцию ahead-of-time (AOT) (опережающую компиляцию), т. е. при установке код из файла *.dex* компилируется в машинный код, хранящийся в файлах с расширением *.oat*, которые заменяют старые файлы *.odex* Dalvik.

лами ZIP, содержащими все ресурсы и метаданные, необходимые приложению для работы. Можно распаковать такой архив, воспользовавшись любой стандартной утилитой, например `unzip`. При распаковке архивного файла выводится следующая информация:

```
> unzip infected.apk

AndroidManifest.xml
classes.dex
resources.arsc
META-INF/
assets/
res/
```

Сначала попытаемся исследовать эти файлы. В частности, файл *AndroidManifest.xml* (**https://developer.android.com/guide/topics/manifest/manifest-intro?hl=ru**), вероятнее всего, может содержать обзорную информацию по этому приложению. Этот файл манифеста обязателен для каждого приложения Android. Он содержит весьма важную информацию о приложении: требуемые права доступа, зависимости от внешних библиотек, компоненты и т. п. Отметим, что нет необходимости объявлять все права доступа, которые приложение использует здесь. Приложения также могут затребовать права доступа во время выполнения, непосредственно перед вызовом функции, которая требует особых прав доступа. (Например, прямо перед активизацией функций фотосъемки открывается диалоговое окно, запрашивающее у пользователя подтверждение права доступа к приложению фотокамеры.) Файл манифеста также содержит следующие объявления:

- activities (операции) – экраны взаимодействия с пользователем;
- services (службы) – классы, работающие в фоновом режиме;
- receivers (приемники широковещательных сообщений) – классы, которые взаимодействуют с событиями системного уровня, такими как СМС или изменения сетевого соединения.

Таким образом, файл манифеста представляет собой великолепный исходный пункт для начала анализа.

Но сразу становится понятно, что почти все распакованные файлы закодированы в некотором бинарном формате. Невозможно даже пытаться просматривать или редактировать эти файлы в таком виде. Для этого необходимы дополнительные инструментальные средства. Apktool (**https://ibotpeaches.github.io/Apktool/documentation/**) – пакет реверс-инженерии для Android, универсальный инструмент, широко используемый для дизассемблирования и декодирования ресурсов, обнаруженных в APK-файлах. После установки этого пакета можно сразу воспользоваться им для распаковки APK в представление, гораздо более удобное для чтения человеком:

```
> apktool decode infected.apk

I: Using Apktool 2.2.2 on infected.apk
I: Loading resource table...
I: Decoding AndroidManifest.xml with resources...
I: Loading resource table from file: <redacted>
I: Regular manifest package...
```

```
I: Decoding file-resources...
I: Decoding values */* XMLs...
I: Baksmaling classes.dex...
I: Copying assets and libs...
I: Copying unknown files...
I: Copying original files...

> cd infected
> ls

AndroidManifest.xml
apktool.yml
assets/
original/
res/
smali/
```

Теперь файл *AndroidManifest.xml* доступен для удобного чтения. Список прав доступа в манифесте является весьма важным основным признаком, который можно использовать для выявления и классификации потенциально опасных приложений. Явные подозрения вызывает запрос приложения более широких прав доступа, чем необходимо в действительности (по нашему мнению). Реально существующее вредоносное приложение с именем пакета *cn.dump.pencil* запрашивает следующий список прав доступа в манифесте:

```
<uses-permission android:name=
    "android.permission.INTERNET"/>
<uses-permission android:name=
    "android.permission.ACCESS_NETWORK_STATE"/>
<uses-permission android:name=
    "android.permission.RECEIVE_BOOT_COMPLETED"/>
<uses-permission android:name=
    "android.permission.READ_PHONE_STATE"/>
<uses-permission android:name=
    "android.permission.ACCESS_COARSE_LOCATION"/>
<uses-permission android:name=
    "android.permission.ACCESS_FINE_LOCATION"/>
<uses-permission android:name=
    "android.permission.ACCESS_WIFI_STATE"/>
<uses-permission android:name=
    "android.permission.WRITE_EXTERNAL_STORAGE"/>
<uses-permission android:name=
    "android.permission.READ_EXTERNAL_STORAGE"/>
<uses-permission android:name=
    "android.permission.MOUNT_UNMOUNT_FILESYSTEMS"/>
<uses-permission android:name=
    "android.permission.GET_TASKS"/>
<uses-permission android:name=
    "android.permission.CHANGE_WIFI_STATE"/>
<uses-permission android:name=
    "android.permission.VIBRATE"/>
<uses-permission android:name=
    "android.permission.SYSTEM_ALERT_WINDOW"/>
```

```
<uses-permission android:name=
    "com.android.launcher.permission.INSTALL_SHORTCUT"/>
<uses-permission android:name=
    "com.android.launcher.permission.UNINSTALL_SHORTCUT"/>
<uses-permission android:name=
    "android.permission.GET_PACKAGE_SIZE"/>
<uses-permission android:name=
    "android.permission.RESTART_PACKAGES"/>
<uses-permission android:name=
    "android.permission.READ_LOGS"/>
<uses-permission android:name=
    "android.permission.WRITE_SETTINGS"/>
<uses-permission android:name=
    "android.permission.CHANGE_NETWORK_STATE"/>
<uses-permission android:name=
    "android.permission.ACCESS_MTK_MMHW"/>
<uses-permission android:name=
    "android.permission.WRITE_SECURE_SETTINGS"/>
```

Если учесть, что это приложение должно предположительно применять стиль карандашного рисунка к фотоизображениям, полученным с камеры, то запрос полного неограниченного доступа в интернет (`android.permisson.INTERNET`) выглядит совершенно необоснованным, так же, как и требование разрешения на вывод окон системных уведомлений (`android.permission.SYSTEM_ALERT_WINDOW`). Согласно официальной документации (**https://developer.android.com/training/permissions/usage-notes?hl=ru**), «приложения крайне редко должны использовать это право доступа; эти окна предназначены для взаимодействия с пользователем на системном уровне». Некоторые другие затребованные права доступа (`WRITE_SECURE_SETTINGS`, `ACCESS_MTK_MMHW`, `READ_LOGS`[1] и др.) явно опасны. Запрашиваемые в манифесте права доступа являются явными признаками, которые можно включить в наш набор признаков. Существует строго определенный набор возможных (допустимых) прав доступа (**https://developer.android.com/guide/topics/permissions/overview#normal_permissions**), которые может затребовать приложение, поэтому кодирование каждого затребованного права доступа как бинарной переменной имеет определенный смысл.

В рассматриваемом пакете заслуживает отдельного внимания сертификат, используемый для подписи приложения. Каждое приложение Android обязательно должно быть подтверждено (подписано) сертификатом, чтобы получить разрешение на выполнение на каком-либо устройстве. Подкаталог *META-INF* в APK содержит ресурсы, которые платформа Android использует для проверки (верификации) целостности и прав владения этого кода, в том числе и сертификат для

[1] Права доступа `WRITE_SECURE_SETTINGS` и `READ_LOGS`, как правило, не должны предоставляться приложениям сторонних авторов, выполняемым на устройствах Android без учетной записи root. `ACCESS_MTK_MMHW` – это привилегия, предоставляющая доступ к специализированной микросхеме FM-радио на некоторых устройствах. Приложения, которые требуют подозрительных или не вполне ясных прав доступа, подобные вышеперечисленным, с высокой вероятностью можно заподозрить во вредоносных действиях. Тем не менее запрос не вполне объяснимых прав доступа не всегда означает, что приложение является вредоносным.

подписи приложения. Apktool помещает подкаталог *META-INF* в корневой каталог пакета. Можно воспользоваться утилитой `openssl` для вывода информации о сертификате в кодировке ASN.1 DER, который представлен файлом **.RSA* в этом подкаталоге[1]:

```
> openssl pkcs7 -in original/META-INF/CERT.RSA -inform DER -print
```

Эта команда выводит подробную информацию о сертификате. Некоторые заслуживающие внимания элементы данных, особенно полезные для определения авторства, находятся в секциях `issuer` и `validity`. В нашем примере можно видеть, что секция `issuer` сертификата не слишком информативна:

```
issuer: CN=sui yun
```

Но срок валидного действия сертификата, по крайней мере, может сообщить, когда было подписано это приложение:

```
notBefore: Nov 16 03:11:34 2015 GMT
notAfter: Mar 19 03:11:34 3015 GMT
```

В некоторых случаях информация `issuer/signer` из сертификата может дать более подробные сведения об авторстве, как показано в следующем примере:

```
Subject
    DN: C=US, ST=California, L=Mountain View, O=Android,
        OU=Android, CN=Android, E=android@android.com
    C: US
    E: android@android.com
    CN: Android
    L: Mountain View
    O: Android
    S: California
    OU: Android
validto: 11:40 PM 09/01/2035
serialnumber: 00B3998086D056CFFA
thumbprint: DF3DAB75FAD679618EF9C9FAFE6F8424AB1DBBFA
validfrom: 11:40 PM 04/15/2008
Issuer
    DN: C=US, ST=California, L=Mountain View, O=Android,
        OU=Android, CN=Android, E=android@android.com
    C: US
    E: android@android.com
    CN: Android
    L: Mountain View
    O: Android
    S: California
    OU: Android
```

Более того, если два приложения имеют одинаковые сертификаты или явно одинаковые подтверждающие подписи, то высока вероятность того, что они созданы одними и теми же авторами. Мы рассмотрим этот случай немного позже.

---

[1] Аргумент `-inform` является сокращением фразы «input format» и позволяет определить входной формат сертификата.

Для получения более подробной информации о приложении необходимо перейти от поверхностного обзора его внутренней структуры к попытке анализа его содержимого.

СТАТИЧЕСКИЙ АНАЛИЗ Статический анализ – это процесс исследования кода приложения без его выполнения. В некоторых случаях, когда доступен удобный для чтения код, например скрипты на языке Python или сниппеты на языке JavaScript, очевидно, что нужно просто внимательно читать такой код и выбирать требуемые признаки, такие как вызовы системного API «с повышенной степенью риска», многократные обращения по сети к внешнему серверу и т. п. В большинстве случаев, как, например, при использовании пакетов приложений Android, необходимо проделать некоторую дополнительную работу по реверс-инженерии приложения. Снова обращаясь к схеме процесса современного выполнения кода, показанной на рис. 4.1, рассмотрим два инструментальных средства из трех указанных на этой схеме: дизассемблер и декомпилятор.

В предыдущем разделе использовался инструмент Apktool для анализа структуры и метаданных файла APK. Если вы обратили внимание на строку `Baksmailing classes.dex...` в выводимой на консоль информации после выполнения команды `apktool decode` для файла *infected.apk*, то, вероятно, сможете предположить, что это такое. Скомпилированный байт-код приложения Android сохраняется в файлах с расширением *.dex* и выполняется виртуальной машиной Dalvik. В большинстве APK скомпилированный байт-код собран в файле с именем *classes.dex*. Baksmali (**https://github.com/jesusfreke/smali**) – это дизассемблер для формата *.dex* (smali – это имя соответствующего ассемблера), который выполняет преобразование объединенного файла *.dex* в исходный код ассемблера smali. Рассмотрим подробнее содержимое подкаталога *smali*, сгенерированного ранее командой `apktool decode`:

```
smali
├── android
│   └── annotation
├── cmn
│   ├── a.smali
│   ├── b.smali
│   ├── ...
├── com
│   ├── android
│   ├── appbrain
│   ├── dumplingsandwich
│   ├── google
│   ├── ...
│   ├── third
│   └── umeng
└── ...
```

Теперь обратимся к содержимому фрагмента класса основной точки входа smali – *smali/com/dumplingsandwich/pencilsketch/MainActivity.smali*:

```
.method public onCreate(Landroid/os/Bundle;)V
    .locals 2
    .param p1, "savedInstanceState"    # Landroid/os/Bundle;
...
```

```
    .line 50
    const/4 v0, 0x1
...
    move-result-object v0
```

Smali – это удобное для чтения человеком представление байт-кода Dalvik. Как и код ассемблера x86, рассмотренный ранее в этой главе, smali может быть трудным для понимания без предварительного изучения. Но в любом случае может оказаться полезной генерация признаков для алгоритма обучения на основе n-грамм[1] инструкций smali. Внимательно изучая код smali, можно заметить некоторые специфические операции, показанные ниже:

```
const-string v0, "http://178.57.217.238:3000"
iget-object v1, p0, Lcom/fanta/services/SocketService;->b:La/a/b/c;
invoke-static {v0, v1}, La/a/b/b;->
    a(Ljava/lang/String;La/a/b/c;)La/a/b/ac;
move-result-object v0
iput-object v0, p0, Lcom/fanta/services/SocketService;->a:La/a/b/ac;
```

В первой строке определяется жестко закодированный IP-адрес для сервера управления и контроля. Во второй строке считывается ссылка на объект из поля экземпляра с размещением `SocketService` в регистре `v1`. В третьей строке вызывается статический метод с передачей в него IP-адреса и ссылки на объект как параметров. Далее результат выполнения этого статического метода перемещается в регистр `v0` и записывается в поле экземпляра `SocketService`. Это одна из форм передачи информации, которую мы пытаемся захватить как элемент признака, сгенерированного с помощью n-грамм кодов операций в формате smali виртуальной машины Dalvik. Например, представление 5-граммы для показанного выше фрагмента кода smali может быть таким:

```
{const-string, iget-object, invoke-static,
    move-result-object, iput-object}
```

Использование n-грамм системных вызовов или кодов операций – весьма перспективное направление для классификации вредоносного ПО[2].

Дизассемблер baksmali способен воспроизводить весь код smali из соответствующего файла .dex, но иногда может выдавать ошеломляющие результаты. Ниже приведено краткое описание некоторых других инструментальных программных сред реверс-инженерии, которые могут ускорить процесс статического анализа:

- radare2 (**https://github.com/radare/radare2**)[3] – широко известная программная среда реверс-инженерии. Это одна из самых простых инструментальных программ для установки и использования, которая предлагает комп-

---

[1] N-грамма – это непрерывная последовательность n элементов, сформированная из более длинной последовательности элементов. Например, 3-граммами для последовательности {1,2,3,4,5} являются {1,2,3}, {2,3,4} и {3,4,5}.

[2] *B. Kang, S.Y. Yerima, K. Mclaughlin and S. Sezer*. N-opcode Analysis for Android Malware Classification and Categorization. Proceedings of the 2016 International Conference on Cyber Security and Protection of Digital Services (2016): 1–7.

[3] Книга документации и руководств по radare2 доступна в онлайновом режиме (**https://radare.gitbooks.io/radare2book/content/**).

лект разнообразных инструментов исследования и анализа, применимых к многочисленным форматам бинарных файлов (не только файлы Android). Radare2 работает на многих операционных системах, например:

– можно воспользоваться командой `rafind2` для поиска по шаблонам байтов в файлах. Это более мощная версия команды ОС Unix `strings`, часто применяемой для поиска и извлечения последовательностей визуально отображаемых символов из бинарных файлов;
– команда `rabin2` предназначена для поиска и вывода свойств бинарного файла. Например, чтобы получить информацию о файле формата *.dex*, нужно выполнить следующую команду:

```
> rabin2 -I classes.dex
...
bintype  class
class    035
lang     dalvik
arch     dalvik
bits     32
machine  Dalvik VM
os       linux
minopsz  1
maxopsz  16
pcalign  0
subsys   any
endian   little
...
```

Для поиска точек входа[1] программы или функции и соответствующих адресов памяти применяется команда:

```
> rabin2 -e classes.dex

[Entrypoints]
vaddr=0x00060fd4 paddr=0x00060fd4 baddr=0x00000000
    laddr=0x00000000 haddr=-1 type=program
```

Для поиска библиотек, импортируемых в выполняемый код, и соответствующих смещениий адресов в таблице компоновки процедур Procedure Linkage Table (PLT)[2] нужно выполнить команду:

```
> rabin2 -i classes.dex

[Imports]
ordinal=000 plt=0x00001943 bind=NONE type=FUNC name=Landroid/app/
    Activity.method.<init>()V
ordinal=001 plt=0x0000194b bind=NONE type=FUNC name=Landroid/app/
```

[1] Точка входа (entry point) – это позиция в коде, в которой управление передается от операционной системы конкретной программе.

[2] PLT – это таблица смещений (offsets)/отображений (mappings), используемых выполняемыми программами для вызова внешних функций и процедур, адреса которых во время компоновки еще не определены. Окончательное определение адреса таких внешних функций выполняется динамическим линкером во время выполнения.

```
    Activity.method.finish()V
ordinal=002 plt=0x00001953 bind=NONE type=FUNC name=Landroid/app/
    Activity.method.getApplicationContext()Landroid/content/Context;
...
```

С помощью radare2 можно выполнять множество других операций, в том числе и в интерактивном сеансе в режиме консоли (командной строки):

```
> r2 classes.dex

# Список всех импортируемых в программу компонентов
[0x00097f44]> iiq

# Список классов и методов
[0x00097f44]> izq
...
```

- Capstone (**http://www.capstone-engine.org/**) – еще одно весьма простое, но мощное инструментальное средство дизассемблирования для многих платформ и архитектур. Основой этой рабочей среды является LLVM, развитая инфраструктура компиляторов, которая может генерировать, оптимизировать и преобразовывать промежуточное представление (intermediate representation – IR) кода, созданного различными компиляторами, например GCC. Хотя кривая обучения Capstone более крутая по сравнению с radare2, Capstone обладает более богатыми функциональными возможностями и обычно больше подходит для автоматизации трудоемких задач дизассемблирования;
- Hex-Rays IDA (**https://www.hex-rays.com/products/ida/**) – самый современный дизассемблер и отладчик, который чаще всего используют профессиональные реверс-инженеры. Эта инструментальная среда предлагает тщательно проработанные комплекты инструментов для выполнения обширного набора функций и операций, но требует приобретения достаточно дорогой лицензии, если необходима самая последняя полная версия этого программного продукта.

Но даже при наличии всех этих инструментов, предназначенных для анализа программ, код smali, возможно, является форматом слишком низкого уровня, чтобы оставаться единственным полезным средством выявления и определения операций с большими областями видимости (действия), которые может выполнять исследуемое приложение. Необходима возможность каким-то образом декомпилировать (decompile) приложение Android в представление более высокого уровня. К счастью, в экосистеме Android существует много инструментальных средств для декомпиляции. Dex2jar (**https://github.com/pxb1988/dex2jar**) – инструментальное средство с открытым исходным кодом для преобразования пакетов APK в файлы JAR, после чего можно воспользоваться декомпилятором JD-GUI (Java Decompiler GUI) (**http://jd.benow.ca/**) для вывода соответствующего исходного кода файлов классов Java, содержащихся в файлах JAR. Но в нашем примере мы воспользуемся другим комплектом инструментов преобразования файлов *.dex* в исходный код Java – JADX (**https://github.com/skylot/jadx**). Можно использовать графический интерфейс JADX-GUI для интерактивного исследования исходного кода приложения Java, как показано на рис. 4.3.

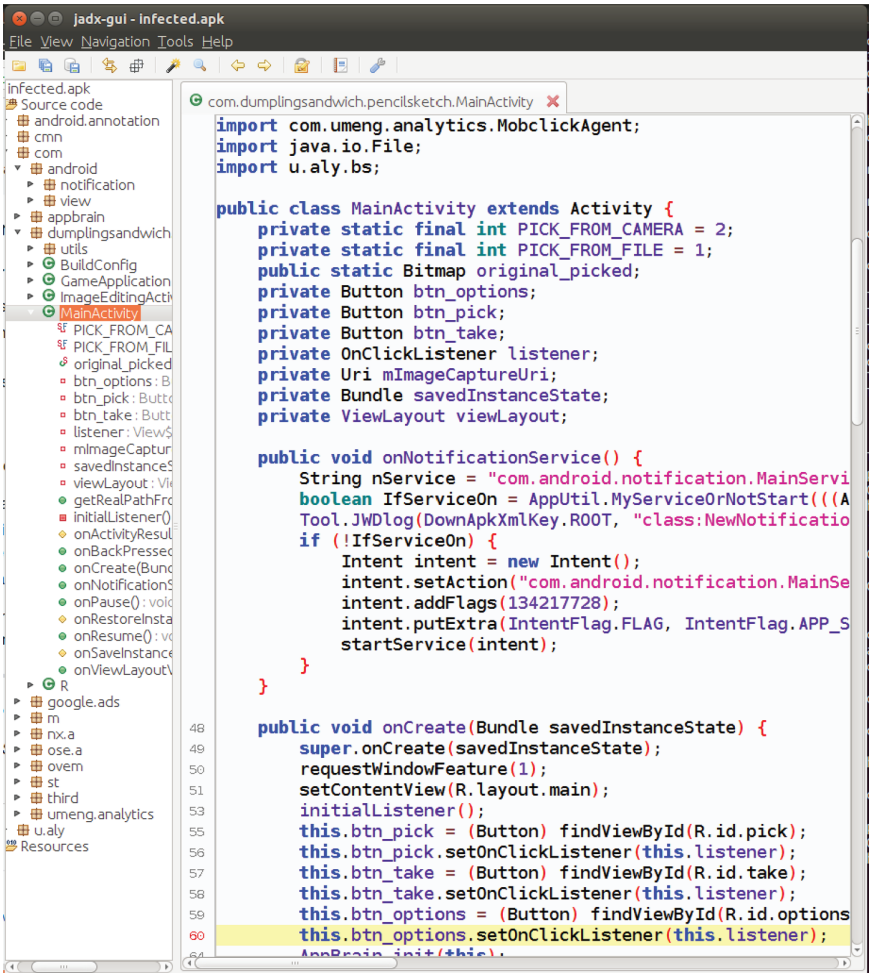


Рис. 4.3 ❖ Декомпилированный исходный код класса Java MainActivity в среде с графическим интерфейсом JADX-GUI

Графический пользовательский интерфейс не подходит для автоматизации генерации кода Java из набора данных в пакете APK, но JADX также предоставляет интерфейс командной строки с возможностью вызова команды `jadx infected.apk`.

Генерация полезных признаков для машинного обучения из исходного кода требует определенных знаний в предметной области, связанной с типичным поведением вредоносного ПО. Вообще говоря, извлекаемые признаки необходимы, для того чтобы получить возможность выявления подозрительных шаблонов кода, жестко закодированных строк, вызовов API и характерных команд, по которым можно предположительно определить поведение, причиняющее вред. Как и при использовании всех ранее рассмотренных методик генерации признаков, можно применить упрощенный метод n-грамм или попытаться выявить признаки, наиболее близкие к уровню детализации, на котором работает человек-аналитик вредоносного ПО.

Даже простое приложение Android может предоставить большой объем кода Java, который необходимо проанализировать для полного понимания того, что делает приложение в целом. При попытке определения вредоносности приложения или обнаружения функциональности отдельных элементов вредоносного ПО аналитики обычно не читают каждую строку кода Java, полученную в результате декомпиляции. Аналитики в определенной степени комбинируют экспертные и практические знания в области типичного поведения вредоносного ПО, чтобы найти характерные особенности программы, которые могут служить подтверждением их решения. Например, вредоносная программа на платформе Android обычно обладает одним или несколькими следующими свойствами:

- использование методов маскировки для скрытия вредоносного кода;
- жестко закодированные строки, ссылающиеся на системные бинарные файлы;
- жестко закодированные IP-адреса серверов управления и контроля или имена хостов;
- проверка, не является ли текущая среда эмулируемой (для предотвращения выполнения в «песочнице»);
- включение ссылок на внешние, скрыто загружаемые или загружаемые обходным путем пакеты APK;
- запрос чрезмерно расширенных прав доступа во время установки или во время выполнения, иногда включая требование административных привилегий;
- содержание только ARM-библиотек для предотвращения выполнения приложения в эмулируемой среде x86;
- следы в виде файлов в непредвиденных и необычных локациях на устройстве;
- изменение собственных легально установленных приложений на устройстве, а также создание или удаление значков для быстрого запуска.

Можно воспользоваться radare2/rafind2 для поиска в рассматриваемом бинарном файле интересующих нас образцов строк, которые могут прямо или косвенно указывать на вредоносное поведение, например такие строки, как `/bin/su`, `http://`, жестко закодированные IP-адреса, ссылки на другие файлы *.apk* и т. п. В интерактивной консоли (интерфейсе командной строки) radare2 выполняется следующая команда[1]:

```
> r2 classes.dex

# Список всех читабельных строк в программе, в которых обнаружен шаблон "bin/su"
[0x00097f44]> izq ~bin/su
0x47d4c 7 7 /bin/su
0x47da8 8 8 /sbin/su
0x47ed5 8 8 /xbin/su

# То же, но с поиском шаблона ".apk"
[0x00097f44]> izq ~.apk
...
```

[1] В проекте radare2 поддерживается краткий оперативный справочник по наиболее часто применяемым командам (**https://github.com/radare/radare2/blob/master/doc/intro.md**).

```
0x72f07 43 43 http://appapk.kemoge.com/appmobi/300010.apk
0x76e17 17 17 magic_encrypt.apk
...
```

Здесь явно обнаружены ссылки на Unix-команду получения привилегий `su` (superuser) и на внешние файлы APK, в том числе на файл с внешнего URL, – это очень подозрительное поведение. С помощью консоли можно продолжить исследование для поиска в коде характерных ссылок на обнаруженные методы и строки, но мы закончим пример на этом и предлагаем для более глубокого изучения обратиться к специальным материалам по этой теме[1].

Упаковка для маскировки

Многие пакеты Android (как обычные, так и вредоносные) используют так называемые программы-упаковщики (packers), или предохранители (protectors), для защиты от реверс-инженерии при помощи маскировки, зашифрования и перенаправления ссылок. Для такой маскировки существует множество вполне законных причин, например защита кода от похищения конкурентами, сжатие в более компактную форму распространяемых объектов и т. д. Разумеется, авторы вредоносных программ для Android быстро освоили эти мощные методики, чтобы затруднить работу специалистов по безопасности по выявлению и изоляции вредоносного ПО. Упаковка файлов приложений не является специализированной методикой для платформы Android – программы-упаковщики существуют и для файлов PE и ELF.

Распаковать приложения Android можно с помощью программ-распаковщиков, например распаковщик Kisskiss (**https://github.com/strazzere/android-unpacker/tree/master/native-unpacker**), который работает с бинарными файлами, упакованными специализированными упаковщиками: Bangcle, APKProtect, LIAPP и Qihoo Android Packers.

АНАЛИЗ ПОВЕДЕНИЯ (ДИНАМИЧЕСКИЙ АНАЛИЗ) Структурный анализ, например исследование метаданных приложения Android, дает лишь весьма ограниченную картину того, что именно делает программа. Статический анализ может лишь теоретически обнаруживать вредоносное поведение с помощью полного исследования всего программного кода, но иногда это требует чрезмерно высоких издержек и затрат ресурсов, особенно при работе с крупными и сложными приложениями. Более того, статический анализ может быть весьма неэффективным, потому что признаки, являющиеся наиболее значимыми сигналами, которые позволяют различать категории бинарных файлов (например, семейство вредоносного ПО, нормальный/вредоносный), чаще всего содержатся лишь в малом фрагменте логики бинарного файла. Анализ 100 блоков кода бинарного файла, чтобы найти единственный блок, содержащий самый значимый признак, – это крайне расточительный способ.

Реальное выполнение программы может стать более эффективным способом генерации полноценных данных. Даже если нет возможности исследовать все

[1] Например, к книге Practical Malware Analysis авторов Michael Sikorski и Andrew Honig (издательство No Starch Press) и Reverse Engineering for Beginners by Dennis Yurichev (**https://beginners.re/**).

существующие в приложении потоки выполнения кода, разные категории бинарных файлов с большой вероятностью будут создавать различные побочные эффекты, которые можно наблюдать и выделять как признаки для классификации.

Для получения точной картины побочных эффектов при выполнении программы проверенным методом является запуск вредоносной программы в «песочнице» для приложений (application sandbox). Выполнение программ в «песочнице» – это специальная методика для изоляции выполнения ненадежного, подозрительного или вредоносного кода с целью защиты хоста от возможного вреда или ущерба.

Самым явно наблюдаемым побочным эффектом при анализе выполняемой вредоносной программы является поведение в сети. Многие вредоносные приложения требуют некоторой формы обмена информацией с внешней средой для получения инструкций от сервера управления и контроля, для передачи похищенных данных или для внедрения и сопровождения нежелательного контента. Наблюдая за взаимодействием с сетевой средой во время выполнения приложения, можно получить динамическую картину («мгновенные снимки») незаконных операций обмена информацией и создать предварительную (приблизительную) сигнатуру исследуемого приложения.

Прежде всего необходимо в среде Android создать «песочницу», в которой будут запускаться приложения. Никогда не выполняйте вредоносные приложения (не важно, являются ли они подозрительными или их вредоносность уже подтверждена) на личных устройствах, где хранятся важные секретные данные. Для запуска таких приложений можно выбрать резервное (неиспользуемое) физическое устройство с ОС Android, но сейчас мы выполним наш пример в эмуляторе Android. Используемый здесь эмулятор создан с помощью менеджера Android Virtual Device (AVD) в среде Android Studio, поддерживающего работу Android 4.4 x86 OS на устройстве Nexus 5 (4.95 1080x1920 xxhdpi). В учебных целях следует обращаться к этому виртуальному устройству по его AVD-имени «pwned». Это неплохое решение – запускать виртуальное устройство Android в одноразовой виртуальной машине, поскольку платформа AVD не обеспечивает изоляцию эмулируемой среды от операционной системы хоста.

Каналом обмена информацией между хостом и эмулятором является Android Debug Bridge (adb) (**https://developer.android.com/studio/command-line/adb**). adb – это инструмент командной строки, который можно использовать для обмена информацией с виртуальным или физическим Android-устройством. Существует несколько различных способов прослушивания сетевого трафика, входящего и исходящего в/из эмулятора (например, старая добрая утилита tcpdump или более многофункциональный Charles proxy (**https://www.charlesproxy.com/**)), но для нашего примера будет использоваться инструмент под названием mitmproxy (**https://mitmproxy.org/**). mitmproxy – это утилита командной строки, предоставляющая интерактивный интерфейс пользователя для исследования и изменения трафика по протоколу HTTP. Для приложений, использующих протокол SSL/TLS, mitmproxy предоставляет собственный корневой (root) сертификат, который можно установить на Android-устройстве для перехвата зашифрованного трафика. Для приложений, в которых правильно реализована фиксация сертификата (certificate pinning) (немногие приложения правильно фиксируют сертификаты),

процесс немного более сложен, но и его можно обмануть[1], пока вы сохраняете управление устройством/эмулятором клиента.

Сначала запускаем mitmproxy в отдельном окне терминала:

```
> mitmproxy
```

Затем запускаем эмулятор. Флаг `-wipe-data` гарантирует, что запуск происходит на чистом дисковом образе эмулятора, а флаг `-http-proxy` перенаправляет трафик через сервер mitmproxy, работающий на `localhost:8080`:

```
> cd <ANDROID-SDK-LOCATION>/tools
> emulator -avd pwned -wipe-data -http-proxy http://localhost:8080
```

После запуска эмулятора необходимо сделать виртуальное устройство видимым для adb. Мы запускаем adb в отдельном окне терминала:

```
> adb devices

List of devices attached
emulator-5554    device
```

Теперь все готово для установки файла APK:

```
> adb install infected.apk

infected.apk: 1 file pushed. 23.3 MB/s (1431126 bytes in 0.059s)
    pkg: /data/local/tmp/infected.apk
Success
```

После возврата в графический интерфейс эмулятора вновь установленное приложение должно без затруднений обнаружиться в секции приложений Android. Для его запуска можно щелкнуть по значку «Pencil Sketch» (рис. 4.4) или запустить это приложение с помощью имени пакет/`MainActivity` (взятого из файла манифеста *AndroidManifest.xml*) через adb:

```
> adb shell am start \
      -n cn.dump.pencil/com.dumplingsandwich.pencilsketch.MainActivity

Starting: Intent { cmp=cn.dump.pencil/
    com.dumplingsandwich.pencilsketch.MainActivity }
```

Рис. 4.4 ❖ Значок вредоносного приложения Pencil Sketch в среде Android

[1] Модуль SSLUnpinning из программной инструментальной среды Xposed Framework позволяет обходить фиксацию сертификата SSL (**https://repo.xposed.info/module/mobi.acpm.sslunpinning**) в приложениях Android. Существуют и другие модули с аналогичной функцией, например JustTrustMe (**https://github.com/fuzion24/justtrustme**).

Теперь мы должны увидеть в графическом интерфейсе эмулятора, что приложение действительно работает (рис. 4.5).

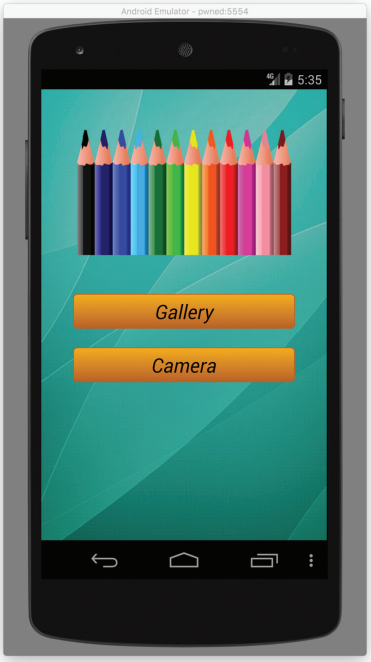


Рис. 4.5. Основной экран вредоносного приложения Pencil Sketch в среде Android

Далее, вернувшись в окно терминала mitmproxy, можно наблюдать в реальном времени перехватываемый трафик, как показано на рис. 4.6[1].

Внимательно изучая выполненные HTTP-запросы, можно сразу же обнаружить некоторый подозрительный трафик:

```
127.0.0.1 GET http://p.appbrain.com/promoted.data?v=11
127.0.0.1 POST http://alog.umeng.com/app_logs
127.0.0.1 POST http://123.158.32.182:24100/
...
127.0.0.1 GET http://218.85.139.168:89/ads_manage/sendAdNewStatus?
user_id=000000000000000&id=-1&
record_type=4&position_type=2&apk_id=993
127.0.0.1 GET http://218.85.139.168:89/ads_manage/getDownloadInfo?
id=0&user_id=000000000000000&ad_class=1
127.0.0.1 POST http://47.88.137.232:7070/
```

[1] Утилиту mitmdump (**https://docs.mitmproxy.org/stable/tools-mitmdump/**) можно использовать для записи перехваченного трафика в файл. Таким образом, можно программно преобразовать перехваченный трафик в формат, более удобный для последующей обработки.

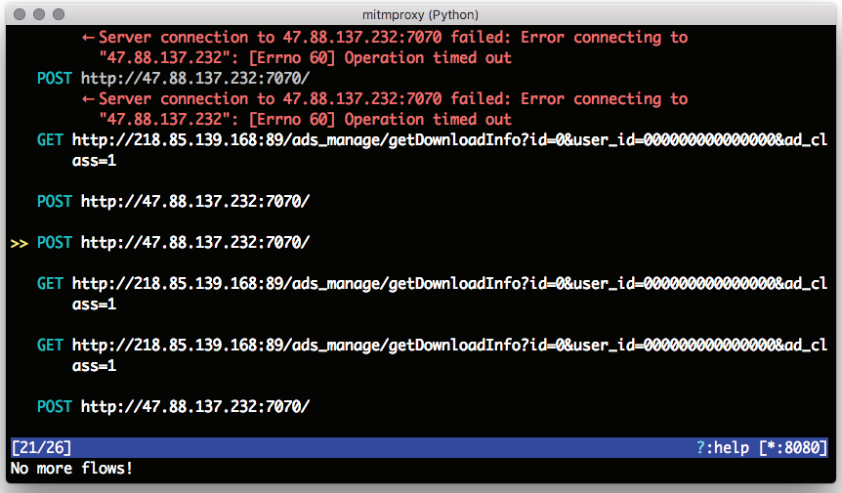


Рис. 4.6 ❖ Интерактивный терминал mitmproxy, отображающий сетевой трафик вредоносного приложения Pencil Sketch в среде Android

Запросы, направленные к хостам `p.appbrain.com` и `alog.umeng.com`, выглядят как безобидный рекламный трафик (Umeng и AppBrain – сети, рекламирующие мобильные приложения), но запросы POST, направляемые по IP-адресам `http://123.158.32.182:24100` и `http://47.88.137.232:7070/`, кажутся весьма подозрительными. Утилита mitmproxy позволяет более тщательно исследовать такие подробности запроса и ответа, как хост, тело запроса POST и т. п., как показано на рис. 4.7.

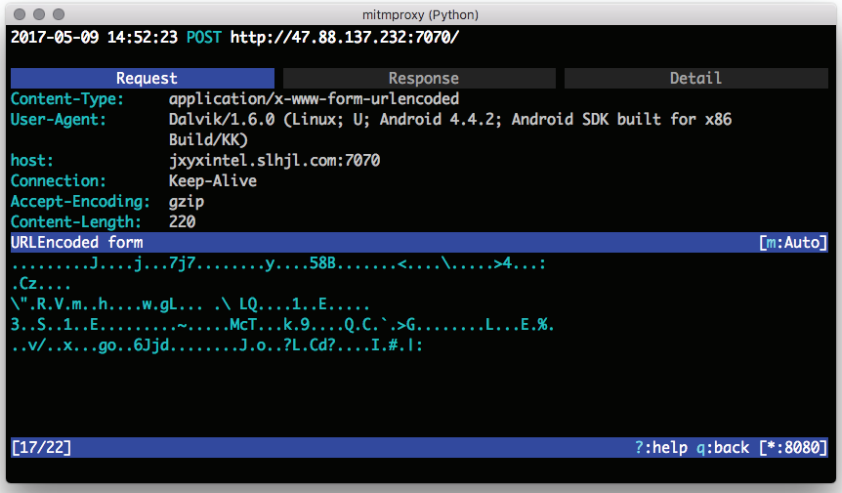


Рис. 4.7 ❖ Интерактивный терминал mitmproxy с подробным выводом содержимого запроса POST к серверу управления и контроля

При исследовании имен хостов и тела запроса кажется вполне вероятным, что хосты `jxyxintel.slhjk.com:7070` и `hzdns.zjnetcom.com:24100` – это серверы управления и контроля. В зависимости от того, насколько новой и актуальной является выборка данных по вредоносному ПО, эти серверы управления и контроля могут оставаться активными или прекратить свою деятельность. В нашем примере на исходящие запросы ответы не получены, поэтому, вероятнее всего, серверы не активны. Но этот факт не должен оказывать существенного влияния на качество извлекаемых признаков.

Кроме профилирования сети, полезно также извлечь некоторые другие поведенческие побочные эффекты приложений Android и использовать их как классификационные признаки:

- последовательность системных вызовов (syscall) во время выполнения приложения – это важный признак, необходимый для успешной классификации вредоносного ПО[1].

  Существует несколько различных способов прослеживания системных вызовов, но наиболее часто применяемым и очевидным является использование модуля `strace` (**https://linux.die.net/man/1/strace**), включенного в большинство современных дистрибутивов Android[2]. Ниже мы кратко рассмотрим, как прослеживаются и анализируются системные вызовы приложения с использованием adb и созданного ранее в нашем примере эмулятора. Приложения Android запускаются посредством ветвления (forking) процесса-демона инициализации приложений Zygote. Поскольку необходимо проследить все системные вызовы приложения с момента запуска его основного процесса, сначала мы запустим `strace` через Zygote, затем выполним поиск (grep) идентификатора процесса приложения в полученных журнальных записях модуля `strace`.

  Предполагая, что целевое приложение уже загружено и установлено на виртуальном устройстве Android, активизируем командную оболочку adb и запускаем `strace` с идентификатором процесса Zygote (следующие команды выполняются из командной оболочки adb)[3]:

---

[1] *Xi Xiao et al.* Identifying Android Malware with System Call Co-occurrence Matrices. Transactions on Emerging Telecommunications Technologies 27 (2016): 675–684.
Marko Dimjasevic et al. Android Malware Detection Based on System Calls. Proceedings of the 2016 ACM International Workshop on Security and Privacy Analytics (2016): 1–8.
Lifan Xu et al. Dynamic Android Malware Classification Using Graph-Based Representations. Proceedings of IEEE 3rd International Conference on Cyber Security and Cloud Computing (2016): 220–231.

[2] Модуль `strace` отсутствует или не работает в некоторых дистрибутивах Android на некоторых платформах. `jtrace` – бесплатное инструментальное средство, заявляемое как «улучшенный модуль `strace` для Android» для вывода специализированной для ОС Android информации, превосходящей по качеству обобщенный и иногда трудный для синтаксического анализа вывод модуля `strace`.

[3] Если вы работаете с более новой версией ОС Android, в которой разрешена подсистема принудительного контроля доступа SELinux, то при попытке запуска `strace` может быть диагностирована ошибка из-за недостаточных прав доступа на выполнение этого модуля. Эту ошибку можно устранить, если установить флаг `androidboot.selinux=permissive` для сборок Android SELinux `-userdebug` и `-eng` или воспользоваться устройством, более лояльным к запуску `strace`.

```
> ps zygote
USER     PID   PPID  VSIZE  RSS     WCHAN    PC         NAME
root      1134  1     707388 46504 ffffffff b766a610 S zygote

> strace -f -p 1134
Process 1134 attached with 4 threads - interrupt to quit
...
```

Затем в другом терминале через adb запускается само приложение:

```
> adb shell am start -n \
      cn.dump.pencil/com.dumplingsandwich.pencilsketch.MainActivity
```

После возврата в окно терминала strace мы должны увидеть сообщения о выполнении некоторых действий:

```
fork(Process 2890 attached
...
[pid  2890] ioctl(35, 0xc0046209, 0xbf90e5c8) = 0
[pid  2890] ioctl(35, 0x40046205, 0xbf90e5cc) = 0
[pid  2890] mmap2(NULL, 1040384, PROT_READ,
MAP_PRIVATE|MAP_NORESERVE, 35, 0) = 0x8c0c4000
...
[pid  2890] clone(Process 2958 attached
...
[pid  2958] access(
"/data/data/cn.dump.pencil/files/3b0b23e7fd0/
f9662419-bd87-43de-ad36-9514578fcd67.zip", F_OK)
= -1 ENOENT (No such file or directory)
...
[pid  2958] write(101, "\4", 1)         = 1
[pid  2958] write(101, "\n", 1)         = 1
```

Должно быть вполне понятно, что здесь показан идентификатор родительского процесса основного приложения, в нашем примере это идентификатор 2890. Отметим, что необходимо также учесть наличие клонов или ответвлений (forks) от родительского процесса приложения. В показанной выше выводимой информации PID 2890 клонирован в другой процесс с идентификатором 2958, который демонстрирует интересующее нас поведение в отношении системных вызовов, которое мы должны с большой вероятностью связать с исследуемым приложением;

- adb предоставляет удобный инструмент командной строки logcat (**https://developer.android.com/studio/command-line/logcat**), который собирает и выводит подробные системные и генерируемые приложением сообщения, ошибки и следы трассировки для всех событий, происходящих в системе. Утилита logcat предназначена для отладки, но иногда она полезна как альтернатива strace при генерации признаков;
- информацию о доступе к файлам и о создании шаблонов можно извлечь из данных о системных вызовах и из результатов трассировки logcat. Эта информация может стать основой важных признаков для классификации вредоносного ПО, поскольку многие вредоносные приложения пытаются получить доступ и выполнить запись в файлы, расположенные в общедоступных или скрытых локациях файловой системы устройства. (Необходимо искать системные вызовы write и access.)

Общепринятым способом генерации характерных признаков по результатам захвата данных о действиях в сети, системных вызовах, трассировке logcat и фактах доступа к файлам является генерация n-грамм последовательностей объектов. Эти последовательности нужно сгенерировать после некоторой предварительной обработки: удаления имен файлов, адресов памяти, излишне специализированных аргументов и т. п. Важно сохранить взаимосвязанные последовательности событий в каждом наборе перехваченных событий с соблюдением баланса случайности (entropy) и стабильности (stability) в генерируемых экземплярах n-грамм. При малом значении n создается меньшее количество возможных неповторяющихся экземпляров n-грамм с меньшей энтропией (случайностью) (следовательно, и с меньшей многозначительностью признаков), но с большей стабильностью, так как приложения, демонстрирующие одинаковое поведение, с большей вероятностью обладают совпадающими наборами характеристик. С другой стороны, при большом значении n мы получаем меньшую стабильность, так как генерируется значительно большее количество неповторяющихся последовательностей экземпляров, но при этом каждый экземпляр формирует гораздо более многозначительный признак. Для выбора правильного значения n требуется несколько практических экспериментов и хорошее понимание взаимосвязи сетевого трафика, системных вызовов и шаблонов доступа к файлам с реальным поведением вредоносной программы. Например, если в программе встречается последовательность из шести системных вызовов для записи некоторых данных, полученных через сетевой сокет, в какой-либо файл, то, вероятнее всего, следует установить для n значение 6.

Динамический анализ – это проверенный общепринятый способ определения характеристик поведения вредоносного ПО. Одного запроса POST, обращенного к подозрительному IP-адресу, возможно, недостаточно, чтобы предъявить обвинение всему приложению, но если эту информацию объединить с подозрительными шаблонами доступа к файлам, последовательностями системных вызовов и запросами на расширение прав доступа, то такое приложение можно классифицировать как вредоносное с высокой степенью уверенности. Машинное обучение хорошо подходит для решения подобных задач, поскольку нечеткое совпадение и малозаметные сходства могут помочь правильно классифицировать намерение и поведение выполняемых файлов.

Недостаток анализа поведения заключается в том, что очень трудно обеспечить полноценный анализ и дать исчерпывающую характеристику всем возможным путям (потокам) выполнения любой программы. Методика фаззинг-тестирования ПО как черного ящика (software fuzzing) помогает найти ошибки в программах, передавая на вход явно некорректные или непредусмотренные данные, но подход с применением принципов фаззинг-тестирования весьма неэффективен при попытке профилирования приложений. Условные операторы и операторы цикла весьма часто встречаются в коде приложения, а конкретные особенные характеристики программы могут проявляться только при редко возникающих, необычных условиях. В качестве примера рассмотрим пример программы на языке Python *secret.py*[1]:

---

[1] Код этого примера можно найти в файле *chapter4/secret.py* (**https://github.com/oreilly-ml-sec/book-resources/blob/master/chapter4/secret.py**) в репозитории кода для данной книги.

```
import sys
if len(sys.argv) != 2 or sys.argv[1] != 's3cretp4ssw0rd':
    print('i am benign!')
else:
    print('i am malicious!')
```

Эта программа проявляет свою «вредоносность» только при выполнении с передачей на вход особенного аргумента – `python secret.py s3cretp4ssw0rd`. Маловероятно, что методики фаззинг-тестирования передадут на вход именно этот аргумент. Данный пример демонстрирует чрезвычайно экстремальный (и надуманный) случай, но аргумент подобного рода может применяться для приложений, требующих некоторых специальных взаимодействий с человеком, прежде чем проявится вредоносное поведение: например, онлайновый вредоносный банковский троян ведет себя обычным образом при запуске, но похищает секретные реквизиты и передает их на удаленный сервер только после успешного выполнения запроса пользователя на вход в банковскую систему. Другой пример: мобильная программа-шантажист, которая проверяет, содержит ли телефонная книга более 20 контактов и имеется ли достаточное количество веб-закладок и записей в журнале посещений, прежде чем начать зашифрование SD-карты. Эти признаки специально предназначены для того, чтобы затруднить использование исследователями вредоносного ПО одноразовых «песочниц» на виртуальных устройствах для профилирования вредоносных программ.

Для генерации признаков, которые способны полностью описать все пространство программы в целом, включая все пути (потоки) выполнения вредоносного и замаскированного кода, необходимо чрезвычайно глубоко исследовать и анализировать непосредственно код такой программы. Это тема следующего подраздела.

ОТЛАДКА Отладчики (например, GDB – бесплатное программное инструментальное средство с открытым исходным кодом, предоставляемое проектом GNU) обычно используются как вспомогательный инструмент при разработке и тестировании (валидации) компьютерных программ, позволяющий проникать в логику приложения и инспектировать промежуточные внутренние состояния исследуемой программы. Но эти инструментальные средства также чрезвычайно полезны на этапе ручного исследования в процессе определения поведения вредоносного ПО. В частности, отладчик позволяет управлять состояниями во время выполнения программы, устанавливая точки прерывания и контрольные точки, условия дампа памяти, и обеспечивает последовательный проход по коду приложения строка за строкой. Этот процесс помогает аналитикам вредоносного ПО быстрее сформировать более полную и ясную картину действий вредоносной программы при помощи наблюдения за тем, что делает программа на каждом шаге выполнения.

В большинстве приложений Android, распространяемых для конечных пользователей, отладка обычно запрещена. Но разрешить режим отладки достаточно просто: необходимо только внести изменения в декодированный файл *AndroidManifest.xml*, добавив атрибут `android:debuggable="true"` в узел этого XML-файла `application`, потом заново упаковать приложение командой `apktool build` и подписать новый созданный файл APK с помощью сертификатов отладки[1]. После этого от-

[1] Существует множество инструментальных утилит независимых авторов, которые с легкостью выполняют процедуру подписи файлов APK для выполнения в режиме отладки, например Uber APK Signer (**https://github.com/patrickfav/uber-apk-signer**).

ладку приложения можно выполнять в штатной официальной среде разработки Android Studio IDE (**https://developer.android.com/studio**) или в более специализированной среде отладки IDA, если у вас имеется лицензия на поддержку отладки для Dalvik. Для отладки на физическом устройстве, где иногда есть возможность получить более реалистичную картину поведения приложения при выполнении, можно также воспользоваться отладчиком более низкого уровня, например KGDB (**https://kgdb.wiki.kernel.org/index.php/Main_Page**)[1].

Отметим, что отладка приложения по своему существу является интерактивным процессом, который практически не поддается автоматизации при работе с неизвестными бинарными файлами. Отладка в процессе извлечения разнообразных информативных признаков для бинарных выполняемых файлов представляет собой важное дополнение к ручной рекогносцировочной работе с целью поиска характерных особенностей программы, к которым в дальнейшем можно применять автоматизированные методики динамического или статического анализа. Например, возможен вариант, в котором большое и сложное игровое приложение Android в основном демонстрирует корректное поведение, но в некоторой непредсказуемой точке выполнения принимает вредоносный код от сервера управления и контроля. Статический анализ, вероятнее всего, не сможет правильно определить такое скрытое поведение, замаскированное сложной логикой приложения. Выполнение приложения в «песочнице» также не гарантирует выявления этого поведения. Если воспользоваться отладчиком для отслеживания поведения программы во внешней сетевой среде и внимательно исследовать все объекты, загружаемые во время выполнения, то мы с большей вероятностью сможем определить момент совершения некорректных действий, проследить такое поведение и выйти непосредственно на соответствующий фрагмент кода. Эта информация дает четкое представление о том, что именно необходимо искать при выполнении статического анализа подобных приложений.

ДИНАМИЧЕСКОЕ ИНСТРУМЕНТИРОВАНИЕ При полном управлении средой времени выполнения приложения можно производить некоторые чрезвычайно действенные операции для воздействия на поведение программы для более удобного извлечения признаков. Динамическое инструментирование (dynamic instrumentation) – это мощная методика изменения приложения или поведения среды времени выполнения при помощи подключения к активно работающим процессам и внедрения собственной логики в контролируемое приложение. Frida (**https://www.frida.re/**) – простое в использовании, динамическое бинарное инструментальное средство с полноценной поддержкой скриптов, с помощью которого можно внедрять код JavaScript в пространство пользователя в приложениях, предназначенных для различных платформ, включая Android, iOS, Windows, macOS и Linux. Можно применять Frida для автоматизации некоторых задач динамического анализа или отладки без трассировки либо фиксации в журналах всех системных вызовов или фактов доступа к сети. Например, можно воспользоваться Frida для регистрации сообщения при любом вызове `open()` в приложении Android:

---

[1] В блоге TrendLabs Security Intelligence Blog представлено подробное руководство (**https://blog.trendmicro.com/trendlabs-security-intelligence/practical-android-debugging-via-kgdb/**) по использованию KGDB.

```
> frida-trace -U -i open com.android.chrome

Uploading data...
open: Auto-generated handler .../linker/open.js
open: Auto-generated handler .../libc.so/open.js
Started tracing 2 functions. Press Ctrl+C to stop.
```

В программной среде Xposed (**https://repo.xposed.info/**) динамическое инструментирование применяется совсем по-другому. Выполняется инструментирование всей виртуальной машины Dalvik с помощью подключения к процессу-демону Zygote, который отвечает за запуск всех приложений и является первоосновой среды времени выполнения Android. Поэтому модуль Xposed может работать в контексте процесса Zygote и выполнять требуемые задачи, такие как обход фиксации сертификата в приложениях (**https://www.bulwarkers.com/2018/02/11/certificate-pinning-bypass/**), подключаясь к общим классам SSL (например, `javax.net.ssl.*`, `org.apache.http.conn.ssl.*` и `okhttp3.*`) для полного обхода всех проверок сертификатов. Модуль `SSLUnpinning`, упоминаемый в предыдущих разделах, представляет собой пример одного из многочисленных модулей, переданных пользователями в репозиторий Xposed Module Repository.

Наряду с методиками защиты от декомпиляции и дизассемблирования приложений Android существуют также некоторые методики противодействия отладке[1] и подключения к внутренним процессам, специально нацеленные на создание трудностей для исследователей, занимающихся отладкой и динамическим инструментированием приложений. В некоторых особенно хитроумных экземплярах вредоносных программ были найдены фрагменты кода, позволяющие обнаруживать широко известные инструментальные средства подключения к внутренним процессам, подобные Xposed, и принудительно завершать их выполнение. В любом случае, потратив время и собственные усилия на решение этой конкретной задачи, почти всегда можно найти способы противодействия и устранения методик маскировки.

КРАТКИЕ ИТОГИ Примеры в этом разделе продемонстрировали широкие возможности инструментальных средств для исследования и анализа бинарных выполняемых файлов. Даже если вы не работаете с конкретными типами выполняемых файлов, рассматриваемых в данной главе, теперь вам известны основные категории доступных инструментальных средств, большинство из которых бесплатно и представляет группу ПО с открытым исходным кодом. Здесь главное внимание было уделено обработке вредоносного ПО на платформе Android, но аналогичные инструменты существуют и для других типов вредоносных программ. Несмотря на то что вам придется искать иные шаблоны поведения, отличающиеся от показанных в этом разделе, всегда полезно знать, что вредоносные программы проявляют себя в попытках получить повышенные системные привилегии, используют неавторизованный сетевой трафик, открывают файлы в странных местах и т. д. Вне зависимости от того, анализируете ли вы вредоносные документы, файлы формата PE или расширения браузера, во всех случаях продолжают

[1] *Haehyun Cho et al.* Anti-Debugging Scheme for Protecting Mobile Apps on Android Platform. The Journal of Supercomputing 72 (2016): 232–246.

применяться основные принципы структурного, статического и динамического анализа для генерации признаков[1].

В этом разделе был показан общий подход к решению задачи генерации признаков, но не обсуждалось, что мы будем делать с этими признаками, какие алгоритмы машинного обучения будут использоваться. Не рассматривалась и значимость каждого сгенерированного признака. Все внимание было сосредоточено на генерации как можно большего количества различных типов описательных признаков из бинарных файлов. Значимость и важность признака в высшей степени зависит от того, чего именно мы хотим добиться с помощью методов машинного обучения. Например, если необходимо классифицировать вредоносное ПО по семействам, то признаки, извлеченные из декомпилированного исходного кода, возможно, будут более важными, чем динамическое поведение в сетевой среде, поскольку авторам вредоносных программ гораздо сложнее заново переписать исходный код, нежели изменить URL или IP-адрес сервера управления и контроля. С другой стороны, если необходимо лишь отделить вредоносные программы от нормальных бинарных файлов, то использование n-грамм системных вызовов, затребованных прав доступа или статический анализ подозрительных строк может быть более эффективным, чем поиск признаков в исходном коде или на уровне ассемблера.

Выбор признаков

В большинстве случаев бездумная загрузка огромного количества признаков в алгоритмы машинного обучения создает ненужный шум и пагубно влияет на точность и эффективность модели. Поэтому важно выбрать только самые важные и значимые признаки для использования в алгоритмах обучения. Этот процесс широко известен как выбор признаков (feature selection). Выбор признаков можно выполнять вручную, на основе экспертных знаний в проблемной области и информации, полученной на этапе исследования данных. Признаки можно выбирать автоматически, используя для этого статистические методы и алгоритмы. Кроме того, существуют методики машинного обучения без учителя, часто применяемые для глубокого обучения (deep learning).

Одним из широко распространенных способов выбора признаков является использование человеческого практического опыта. Люди-эксперты могут обеспечить процесс руководства моделями машинного обучения, который проявляется главным образом в форме добытых вручную признаков, считающихся наиболее важными элементами информации, используемыми в процессе обучения человека. Например, во время тренировки механизма бинарной классификации птиц и млекопитающих огромное количество разнообразных признаков может быть сгенерировано по каждому элементу (например, млекопитающее): размер, масса тела, происхождение, количество ног и т. п. Но любой ребенок сможет назвать один самый важный признак, отличающий птиц от млекопитающих, – наличие перьев. Без участия человека в этом смысле алгоритм машинного обучения может сохранять способность находить сложную границу решения в многомерном

[1] Отладка и динамическое инструментирование могут быть абсолютно неприменимыми в тех случаях, когда существующие инструменты несовершенны, например могут полностью отсутствовать отладчики для исследования вредоносных файлов формата PDF.

пространстве и обеспечивать высокую точность классификации. Но модель, выбор признаков для которой выполняется с участием человека, будет гораздо более простой и эффективной.

Статистически управляемые алгоритмы выбора признаков – это широко применяемые методики снижения размерности наборов данных как с предварительным выбором признаков вручную, так и без оного. Рассмотрим подробнее эти методики и классифицируем их по нескольким семействам (категориям):

- одномерный анализ (univariate analysis): интуитивно понятной и хорошо обобщаемой методикой выбора признака является исследование того, насколько хорошо работает модель, если в качестве входных данных ей предлагается только этот признак. С помощью итеративно выполняемых одномерных статистических тестов по каждому отдельному признаку можно вывести относительную оценку того, насколько хорошо каждый признак соответствует распределению меток в тренировочном наборе. Библиотека scikit-learn предлагает несколько методов одномерного анализа, которые выбирают только наиболее полно описанные признаки в исследуемом наборе данных. Например, класс `sklearn.feature_selection.SelectKBest` (**https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectKBest.html**) сохраняет лишь признаки с максимальными оценками, принимая как аргумент функцию одномерной статистической оценки соответствия, например критерий хи-квадрат (**https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.chi2.html**) или ANOVA (**https://scikit-learn.org/stable/auto_examples/svm/plot_svm_anova.html**) (с использованием F-значения). Часто применяется такой метод выбора признаков с помощью одномерного анализа, в котором удаляются признаки, крайне незначительно изменяющиеся в различных элементах выборки. Если признак имеет одно и то же значение в 99 % элементах выборки, то, вероятнее всего, его включение в процесс анализа не принесет особой пользы. Класс `sklearn.feature_selection.VarianceThreshold` (**https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.VarianceThreshold.html**) позволяет определить минимальный порог изменяемости для признаков, которые предполагается использовать в дальнейшем;
- рекурсивное исключение признаков (recursive feature elimination): действуя с противоположного направления, такие методы, как `sklearn.feature_selection.RFE` (**https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.RFE.html**), начинают с обработки полного набора признаков и рекурсивно рассматривают постоянно уменьшающиеся подмножества признаков с анализом того, как исключение признаков влияет на точность тренировки модели оценки, предложенной исследователем;
- неявное представление признаков (latent feature representations): такие методы, как сингулярное разложение (Singular Value Decomposition – SVD) и метод главных компонентов (Principal Component Analysis – PCA), выполняют преобразование данных с высокой размерностью в пространства данных с более низкой размерностью. Эти алгоритмы предназначены для минимизации потерь информации при сокращении количества признаков, необходимых для эффективной работы моделей машинного обучения. Класс `sklearn.decomposition.PCA` (**https://scikit-learn.org/stable/modules/generat-**

ed/sklearn.decomposition.PCA.html) извлекает главные компоненты из входных признаков, затем исключает все, кроме компонентов, находящихся на самой вершине иерархии, которые максимизируют дисперсию, формируемую для этого набора данных. Отметим, что эти методы с технической точки зрения не выполняют «выбор признаков», поскольку фактически не выбирают признаки из исходного набора. Вместо этого они на выходе выдают признаки, являющиеся результатом матричных преобразований, и не обязательно соответствуют какому-либо из исходных измерений признаков;

- классификация признаков в зависимости от конкретной модели (model-specific feature ranking): когда алгоритмы машинного обучения применяются к некоторому набору данных, модели итоговой оценки иногда могут быть выражены в форме символьных комбинаций входных признаков. Например, для модели линейной регрессии, в которой прогнозируется значение Y на основе трехмерного набора данных (в котором признаки обозначены как x_a, x_b и x_c), модель регрессии может быть представлена следующей формулой (без учета отклонений):

  $$Y = W_a x_a + W_b x_b + W_c x_c.$$

  После этапа тренировки коэффициентам (весам) W_a, W_b и W_c будут присвоены некоторые значения, например:

  $$Y = 4{,}96 x_a + 2{,}64 x_b + 0{,}02 x_c.$$

  Даже в этом простейшем примере можно ясно видеть, что признаки x_a и x_b имеют более высокие весовые коэффициенты, чем признак x_c. Предполагая, что эти признаки нормализованы соответствующим образом (т. е. их значения являются сравнимыми величинами), можно исключить признак x_c, зная, что он не будет оказывать существенного влияния на работу модели регрессии. Методы регуляризации, использующие метрику (норму) L_1, по самой природе собственного процесса регуляризации содержат много нулевых оценочных коэффициентов. Использование класса `sklearn.feature_selection.SelectFromModel` (**https://scikit-learn.org/stable/modules/generated/sklearn.feature_selection.SelectFromModel.html**) для исключения подобных признаков из набора данных является хорошей практической методикой, позволяющей получить более компактную модель оценки с высокой эффективностью. Класс `SelectFromModel` также успешно применяется и для других моделей машинного обучения, в том числе и для моделей на основе деревьев[1]. Модели классификации на основе деревьев могут генерировать метрику для относительной важности каждого входного признака, так как некоторые входные признаки способны более точно распределять тренировочные данные по правильным меткам классов по сравнению с другими признаками[2].

---

[1] *Chotirat Ann Ratanamahatana and Dimitrios Gunopulos.* Scaling Up the Naive Bayesian Classifier: Using Decision Trees for Feature Selection. University of California (2002).

[2] Хороший пример использования класса метаоценки `SelectFromModel` в методе `sklearn.ensemble.ExtraTreesClassifier` для выбора только самых важных признаков для классификации приведен в документации на библиотеку scikit-learn (**https://scikit-learn.org/stable/modules/feature_selection.html#tree-based-feature-selection**).

Обучение признакам без учителя и глубокое обучение

Существует класс алгоритмов глубоких нейронных сетей, которые могут автоматически обучаться представлениям признаков иногда даже по данным без меток. Такие алгоритмы предлагают кажущуюся невероятной возможность существенного сокращения времени, затрачиваемого на конструирование признаков, – обычно это один из этапов машинного обучения, требующий самых больших затрат времени. В частности, нейронная сеть-автокодировщик (autoencoder neural network) (**http://ufldl.stanford.edu/wiki/index.php/Stacked_Autoencoders**) – это алгоритм обучения без учителя, применяющий методику тренировки с обратным распространением (ошибок) для создания сети, которая обучается воспроизведению (репликации) входного элемента выборки в выходном слое. Эта задача может показаться тривиальной, но, создавая труднопроходимый скрытый слой в такой сети, мы реализуем ее специализированную тренировку для обучения эффективному способу сжатия и воспроизведения входных данных при минимизации потерь (т. е. различий) между входом и выходом. На рис. 4.8 показан простой автокодировщик с одним скрытым слоем.

В этом примере сеть пытается научиться без учителя, как сжимать информацию, требуемую для воспроизведения (в выходном слое), из пятимерного входа в трехмерный набор признаков в скрытом слое.

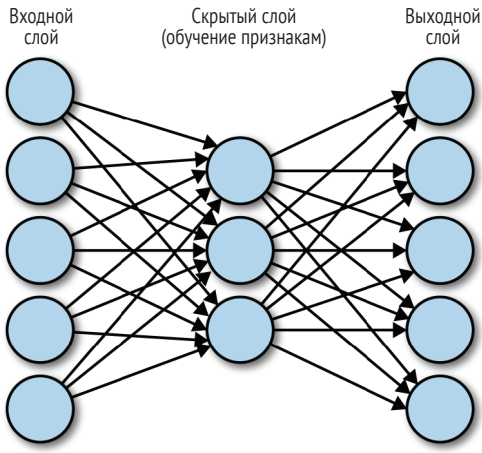


Рис. 4.8 ❖ Полностью связанная сеть-автокодировщик 5-3-5

Данные для ввода в нейронные сети, как правило, отличаются от входных данных обычных моделей машинного обучения. Такие алгоритмы, как леса случайных деревьев и метод опорных векторов, лучше всего работают на правильно подготовленных наборах признаков, а глубокие нейронные сети дают наилучший результат при работе с максимально возможным разнообразием необработанных заранее признаков, генерируемых из исходных данных. Вместо тщательной подготовки и конструирования признаков мы позволяем алгоритму обучения самому создавать признаки.

Глубокое обучение, или обучение признакам без учителя, – это область интенсивных научных исследований, которая существенно отличается от других методик машинного обучения. Но даже при наличии большого объема теоретических работ в этой области данная методика пока еще не получила широкого распространения на практике. В этой книге мы не рассматриваем какие-либо подробности глубокого обучения. Желающим более подробно ознакомиться с этой темой можно порекомендовать специализированные материалы и публикации, например книгу Deep Learning Иэна Гудфеллоу (Ian Goodfellow), Йошуа Бениго (Yoshua Benigo) и Аарона Курвиля (Aaron Courville) (издательство MIT Press).

От признаков к классификации

Автоматизация процесса извлечения признаков – это отдельная задача. Для генерации набора признаков из необработанных исходных данных, таких как бинарные файлы, необходим соответствующий уровень знаний и практических навыков программирования и написания скриптов. Здесь мы не будем подробно рассматривать подробности этого процесса, поскольку задача в высокой степени зависит от типа обрабатываемых данных и от цели классификации. Вместо этого мы приведем несколько превосходных примеров, доступных для ознакомления и тестирования проектов с открытым исходным кодом, которыми вы можете воспользоваться для дальнейшего освоения данной темы:

- youarespecial (**https://github.com/endgameinc/youarespecial**), автор Хайрем Андерсон (Hyrum Anderson), Endgame Inc. – к коду в этом репозитории прилагается информационное сообщение Хайрема Андерсона. Это хороший пример реализации извлечения признаков и глубокого обучения для классификации вредоносного ПО по бинарным файлам формата PE ОС Windows. Например, класс `PEFeatureExtractor` извлекает полный набор статических признаков, включая «сырые признаки», для которых не требуется синтаксический анализ PE-файла, в том числе:
  - гистограмма значений байтов – гистограмма распределения значений байтов в бинарном файле;
  - гистограмма энтропии байтов – двумерная гистограмма энтропии байтов, аппроксимирующая «объединенную вероятность содержания значения в байте и локальную энтропию»[1];
  - строки: массив статистических данных о строках, извлекаемых из исходного потока байтов, определяется пятью и более последовательными символами, имеющими ASCII-значения между 0x20 (пробел) и 0x7f (del), или специальными строками, например C:\, HKEY_, http://, и содержит такие характеристики, как количество строк, средняя длина строки, количество путей C:\, экземпляров URL, ключей реестра, а также гистограмма распределения символов;

[1] Эта функциональная возможность была предложена в разделе 2.1.1 «Deep Neural Network Based Malware Detection Using Two-Dimensional Binary Program Features», Джошуа Сакс (Joshua Saxe) и Константин Берлин (Konstantin Berlin) (Материалы конференции MALWARE 2015).

– некоторые «синтаксически анализируемые характеристики», например:
  - общая информация о файле – подробности высокого уровня о файле формата PE, например скомпилирован ли он с отладочными символами, количество экспортируемых/импортируемых функций и т. д.;
  - информация о заголовке файла – подробности, которые можно получить из секции заголовка PE-файла, относящиеся к компьютеру, архитектуре, ОС, линкеру и т. п.;
  - информация о секциях бинарного файла – имена, размеры, энтропия;
  - информация об импорте – информация об импортируемых библиотеках и функциях, которые могут использоваться в исследуемом PE-файле;
  - информация об экспорте – информация о функциях, экспортируемых из исследуемого PE-файла.

Хеширование признаков

Отметим, что для некоторых признаков в проекте youarespecial, например для информации о секциях файла, используется хеширование признаков (feature hashing) для кодирования извлекаемых признаков в векторизованной форме. Эта методика также известна под названием «прием с хешированием» (hashing trick), поскольку представляет собой быстрый и компактный способ векторизации признаков. В библиотеке scikit-learn реализован метод хеширования признаков, представленный классом `sklearn.feature_extraction.FeatureHasher` (**https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.FeatureHasher.html**).

Хеширование признаков осуществляется с помощью хеш-функции, применяемой к имени признака, а полученное хеш-значение используется как индекс вектора для хранения соответствующего значения признака. Сохраняя эти значения под соответствующими индексами в разреженной матрице (sparse matrix) и управляя количеством неповторяющихся значений, которые может сгенерировать хеш-функция, можно установить жесткий предел размера матрицы признаков. Например, рассмотрим следующий список признаков и соответствующих им значений:

Таблица 4.1

Имя признака	Значение признака
FUNCTION_READFILE_CNT_CODE	67
FUNCTION_READFILE_ENTROPY	0.33
FUNCTION_READFILE_NUM_IO	453
FUNCTION_VALIDATECONFIG_CNT_CODE	54
FUNCTION_VALIDATECONFIG_ENTROPY	0.97
FUNCTION_VALIDATECONFIG_NUM_IO	24587

Применяя методику хеширования признаков, мы передаем имена признаков в хеш-функцию, которая возвращает хеш-значение от 0 до 99, и получаем следующий результат:

Таблица 4.2

Имя признака	Хеш-значение
FUNCTION_READFILE_CNT_CODE	32
FUNCTION_READFILE_ENTROPY	64
FUNCTION_READFILE_NUM_IO	39
FUNCTION_VALIDATECONFIG_CNT_CODE	90
FUNCTION_VALIDATECONFIG_ENTROPY	33
FUNCTION_VALIDATECONFIG_NUM_IO	4

Затем признаки кодируются в форме разреженного вектора (матрицы) с использованием хеш-значений признаков как индексов для хранения соответствующих значений признаков:

```
index  4      ...  32  33    ...  39   ...  64    ...  90
value  24587  -    67  0.97  -    453  -    0.33  -    54
```

Отметим, что эта методика чувствительна к хеш-коллизиям, возникающим в случаях, когда два или более значений признаков индексируются в одну и ту же позицию, что приводит к перезаписи значения, потере информации и сохранению неправильного значения для признака. Коллизии такого рода могут стать причиной повреждений данных в различной степени, в зависимости от размера набора данных, а также от того, были ли значения нормализованы. Но существует несколько решений по устранению эффектов хеш-коллизий, например использование дополнительных хеш-функций[1].

Хеширование признаков обосновано при обработке большого и/или потенциально не связанного количества признаков в наборе данных. Чаще всего это задачи классификации текстов с созданием матриц терминов из документов. Кроме того, эта методика достаточно часто применяется для обработки особо сложных исходных данных, таких как бинарные файлы, когда возможно извлечение сотен тысяч признаков.

Ниже приведен небольшой фрагмент кода, демонстрирующий простой вариант применения класса PEFeatureExtractor для генерации списка важных признаков, описывающих файл формата PE:

```
import os
from pefeatures import PEFeatureExtractor

extractor = PEFeatureExtractor()
bytez = open(
    'VirusShare_00233/VirusShare_fff8d8b91ec865ebe4a960a0ad3c470d,
    'rb').read()
feature_vector = extractor.extract(bytez)

print(feature_vector)
> [  1.02268000e+05   3.94473345e-01   1.79919427e-03 ...,
     0.00000000e+00   0.00000000e+00   0.00000000e+00]
```

[1] *Kilian Weinberger, Anirban Dasgupta, John Langford, Alex Smola and Josh Attenberg*. Feature Hashing for Large Scale Multitask Learning. Proc. ICML (2009).

- ApkFile (**https://github.com/calebfenton/apkfile**), автор Калеб Фентон (Caleb Fenton), – это библиотека Java, обеспечивающая для файлов APK обработку, аналогичную той, которую класс `PEFeatureExtractor` обеспечивает для файлов PE. Библиотека предоставляет «надежный способ инспектирования опасных экземпляров вредоносного ПО» с генерацией в удобном виде статической информации из различных уровней, включая манифест, структуру файла *.dex* и декомпилированный код. Как и в показанном выше примере, API библиотеки поддерживает простое написание скриптов и выполнение автоматизации извлечения вектора признаков из большого комплекта файлов APK;
- LIEF (**https://github.com/lief-project/lief**) от Quarkslab: класс `PEFeatureExtractor` использует LIEF (Library to Instrument Executable Formats) как внутренний механизм для синтаксического анализа файлов PE. LIEF – это весьма мощная и гибкая библиотека, поддерживающая синтаксический анализ, изменение и составление описаний файлов в форматах ELF, PE и MachO. С минимальными трудозатратами можно отредактировать `PEFeatureExtractor` и создать собственный `ELFFeatureExtractor`, просто заменив используемый API `lief.PE` на API `lief.ELF`, также предоставляемый библиотекой.

Как получить образцы и метки вредоносного программного обеспечения

Получение образцов бинарных выполняемых файлов для тренировки классификаторов может стать весьма непростой задачей. Джон Сеймур (John Seymour) (**@_delta_zero**) собрал подробнейший список наборов данных из вредоносных программ, которым можно воспользоваться для тренировки классификаторов (**https://sector.ca/wp-content/uploads/presentations16/Seymour.pdf**):

- VirusTotal (**https://www.virustotal.com/#/home/upload**) – широко известный источник образцов вредоносного ПО и методик структурного анализа. Также предлагает большой набор продуктов для анализа вирусов по запросу с API для получения информации о компонентах вредоносного ПО на основе хеш-значения бинарного файла. Для доступа к большинству сервисов сайта требуется секретный ключ API (его необходимо приобрести), а образцы и метки предоставляются под защитными лицензиями;
- Malware-Traffic-Analysis.net (**http://malware-traffic-analysis.net/**) – небольшой набор данных, содержащий 600 тщательно проанализированных образцов вредоносного ПО и файлов PCAP. Размер набора данных слишком мал для тренировки классификатора машинного обучения, но это хороший ресурс для экспериментов с признаками и ознакомления с вредоносным ПО;
- VirusShare.com (**https://virusshare.com/**) – огромный (около 30 миллионов образцов на момент написания книги) и свободный (бесплатный) репозиторий вредоносного ПО, предоставляющий работающие образцы (распространяемые через торрент-сервис) для исследователей в области безопасности. Доступ к образцам предоставляется по приглашению (invitation), но запрос приглашения можно сделать по электронной почте, обратившись к администраторам сайта. Джон Сеймур и MLSec Project (**https://github.com/mlsecproject**) вносят наибольший вклад в постоянно пополняющийся набор меток для образцов VirusShare с подробной информацией о каждом файле.

Эти метки публикуются на объединенном ресурсе, на который можно получить доступ по ссылке с SecRepo.com (**http://www.secrepo.com/**) (Samples of Security Related Data – еще один полезный ресурс, содержащий наборы данных для обеспечения безопасности);

- VX Heaven (**http://83.133.184.251/virensimulation.org/**) – известный в академических научных кругах набор данных, содержащий около 270 000 образцов вредоносного ПО с классификацией по почти 40 классам. Образцы не обновлялись около 10 лет, поэтому не следует рассчитывать на предоставление характеристик современного вредоносного ПО;
- Kaggle и Microsoft в 2015 г. организовали проект Malware Classification-Challenge (**https://www.kaggle.com/c/malware-classification**) и создали около 10 000 образцов вредоносных файлов PE (заголовки PE-файлов удалены, поэтому образцы неработоспособны), доступных участникам конкурса. На момент написания книги образцы оставались доступными, но условия Kaggle ограничивают использование этого набора данных исключительно для участников данного конкретного конкурса.

Существует множество нюансов и тонкостей при создании хорошего набора данных для классификации вредоносного ПО, которые не относятся к теме данной книги. Для более глубокого изучения материалов по этой теме можно рекомендовать ресурсы, соданные и поддерживаемые Джоном Сеймуром (John Seymour), Хайремом Андерсоном (Hyrum Anderson), Джошуа Саксом (Joshua Saxe) и Константином Берлиным (Konstantin Berlin).

Резюме

Конструирование признаков – один из наиболее важных, но и наиболее трудных и требующих больших затрат времени этапов разработки решений задач машинного обучения. Для конструирования действительно значимых признаков из необработанных исходных данных недостаточно быть только специалистом в области датологии или машинного обучения. Обширные экспертные знания в прикладной области являются весьма полезным и даже обязательным требованием, которое может существенно помочь или стать препятствием при попытках разработки решений на основе машинного обучения для конкретной задачи. В области обеспечения безопасности, где многие прикладные направления могут извлечь пользу из методов машинного обучения, каждое направление требует отличных от других экспертных знаний, которыми должен обладать специалист. В этой главе было проанализировано конструирование признаков для приложений, обеспечивающих безопасность и защиту на основе методов машинного обучения. Подробно рассматривая процесс анализа бинарных вредоносных файлов и реверс-инженерию как практический пример извлечения признаков, мы разработали обобщенный набор принципов и правил, которые можно применять к другим типам приложений, использующих машинное обучение для обеспечения безопасности.

В главе 5 мы рассмотрим, как можно использовать набор извлеченных признаков для выполнения классификации и кластеризации.

Глава 5

Анализ сетевого трафика

Наиболее вероятный способ, которым атакующие могут проникнуть в вашу инфраструктуру, – это путь через сетевую среду. Обеспечение безопасности сети (network security) представляет собой обширную практическую область деятельности по защите компьютерных сетей и конечных пунктов сетевого доступа от вредоносных действий, некорректного использования и критических отказов (аварий)[1]. Сетевые защитные экраны (firewalls) являются, вероятно, самыми широко известными системами защиты сетей, применяющими стратегии доступа и фильтрацию неавторизованного трафика между объектами в сетевой среде. Но обеспечение защиты сети не ограничивается только лишь одними сетевыми экранами.

В этой главе мы рассмотрим методики классификации сетевого трафика. Сначала будет сформирована модель защиты сети как основа для дальнейшего обсуждения. Затем будут исследоваться более глубоко задачи обеспечения безопасности сети, при решении которых уже удалось получить преимущества от разработок в области искусственного интеллекта и машинного обучения. Во второй части главы приводится пример практического использования машинного обучения для поиска шаблонов и обнаружения взаимосвязей (корреляций) в сетевых данных. Используя науку о данных как исследовательский инструмент, мы подробно рассматриваем, как применять методы классификации к сложным наборам данных для обнаружения атак в сетевой среде.

Тематика обсуждения обеспечения безопасности сети здесь ограничена уровнем передачи информации на основе сетевых пакетов. При пакетной передаче поток данных разделяется на более мелкие элементы, каждый из которых содержит некоторые метаданные об источнике передачи, о цели, а также собственно полезное содержимое. Каждый пакет передается через сетевой уровень и форматируется по правилам соответствующего протокола на транспортном уровне

[1] В рамках этой книги мы ограничиваемся определением термина «сеть», «сетевая среда» как системы соединений, позволяющей передавать данные между соединяемыми узлами. Существует огромный диапазон носителей и сред передачи данных, протоколов обмена информацией, сетевых топологий и конфигураций инфраструктуры, которые влияют на любое подробное обсуждение проблем сетевой безопасности, но здесь мы не рассматриваем все подробности. При обсуждении темы данной главы мы используем несколько важных обобщенных вариантов обеспечения сетевой безопасности. Вы сможете применить на практике многие обобщенные концепции и методики, рассматриваемые здесь, в конкретных ситуациях и вариантах, даже если существуют некоторые различия в используемых средах и используемых протоколах.

с восстановлением информационного потока из отдельных пакетов, выполняемом на уровне сеанса или на более высоком уровне. Защита сетевого, транспортного и сеансового уровней (уровни 3, 4 и 5 в модели OSI соответственно) является основной темой данной главы.

Модель OSI

На протяжении всей текущей главы мы ссылаемся на различные части типового сетевого стека, используя при этом общеизвестную модель OSI (Open Systems Interconnection) (**https://ru.wikipedia.org/wiki/Сетевая_модель_OSI**). Модель OSI содержит семь уровней (рис. 5.1):

- 1 – физический уровень (physical layer) – выполняется преобразование цифровых данных в электрические или оптические сигналы, которые могут быть переданы по сети, а также обратное преобразование сигналов, полученных по сети, в цифровые данные;
- 2 – канальный уровень (data link layer) – управление передачей данных между связанными узлами в физической сети;
- 3 – сетевой уровень (network layer) – выполняется маршрутизация пакетов и осуществляется управление потоком между двумя пунктами в сети;
- 4 – транспортный уровень (transport layer) – обеспечение обмена информацией между хостами (host-to-host), а также качества и надежности передачи данных;
- 5 – сеансовый уровень (session layer) – обеспечение инициализации, поддержки и завершения сеансов между процессами приложений;
- 6 – уровень представления (presentation layer) – выполняется преобразование бинарных данных в форматы, доступные для приложений;
- 7 – прикладной уровень (application layer) – обеспечение взаимодействия пользовательских приложений с сетью.

Рис. 5.1 ❖ Сетевой стек, соответствующий семиуровневой модели OSI

Как уже отмечалось в предыдущих главах, при атаках сети используются главным образом как средство достижения более конкретных целей: шпионаж, похищение данных, нанесение ущерба и т. п. Из-за сложности и повсеместного распространения различных типов связи между компьютерными системами атакующие

могут постоянно искать пути доступа даже в системы, спроектированные самым тщательным образом. Следовательно, мы должны сосредоточиться на тех вариантах, в которых предполагается, что атакующий уже преодолел внешний периметр и активно действует внутри защищаемой сетевой среды.

Теория защиты сетей

Модель защиты сетей сложна из-за широкого диапазона поверхностей атак и векторов потенциальных угроз. Как и при защите любой сложной системы, администраторы должны противостоять атакующим на многих направлениях и не полагаться на надежность какого-либо одного компонента решения.

Управление доступом и аутентификация

Взаимодействие клиентов с сетью начинается с уровня управления доступом. Управление доступом – это форма авторизации, позволяющая администратору управлять, определяя, какие пользователи, роли или хосты в организации могут получить доступ к каждому сегменту сети. Сетевые защитные экраны (firewalls) являются средствами управления доступом, применяя предварительно определенные стратегии и правила, определяющие, как хосты в сети могут обмениваться информацией друг с другом. В ядро Linux включен встроенный сетевой экран iptables (**https://netfilter.org/**), применяющий набор правил уровня протокола IP, который определяет возможности установления входящих и исходящих соединений для любого хоста, с конфигурированием в командной строке. Например, системный администратор, которому необходимо так сконфигурировать iptables, чтобы разрешить только входящие соединения по протоколу Secure Shell (SSH) (через порт 22) из конкретно заданной подсети 192.168.100.0/24[1], может выполнить следующие команды:

```
# Принимать (ACCEPT) входящие TCP-соединения из подсети 192.168.100.0/24 на порт 22
iptables --append INPUT --protocol tcp --source 192.168.100.0/24
         --dport 22 --jump ACCEPT

# Запретить (DROP) все прочие входящие TCP-соединения на порт 22
iptables --append INPUT --protocol tcp --dport 22 --jump DROP
```

Подобно дверным замкам в здании, стратегии управления доступом важны для гарантии того, что только конечный пользователь с явно предоставленными ему правами доступа (аналог физического ключа для запертой двери) может войти в сеть. Но точно так же, как замок может быть взломан, открыт с помощью украденного ключа, или вор проникает через окно, любую систему пассивного управления доступом можно обойти. Атакующий, который захватывает управление сервером в подсети 192.169.100.0/24, получает доступ к этому серверу, потому что метод пассивной аутентификации полагается на единственную характеристику – исходный пункт соединения, – когда решает, разрешить или запретить доступ в этом конкретном случае.

Методы активной аутентификации (active authentication) собирают больше информации о клиенте, пытающемся установить соединение, часто используя при

[1] Для обозначения IP-подсетей здесь используется нотация CIDR (**https://tools.ietf.org/html/rfc4632#section-3.1**).

этом криптографические методики, секретные данные или распределенные механизмы для более надежной идентификации клиента. Например, в дополнение к исходному пункту соединения как идентификационному сигналу системный администратор может потребовать ключ SSH и/или дополнительные регистрационные данные для аутентификации, чтобы разрешить запрашиваемое соединение. В некоторых случаях многофакторная аутентификация (multifactor authentication – MFA) может быть подходящим методом для создания препятствия, мешающего атакующим прорваться внутрь. Многофакторная аутентификация разделяет процедуру аутентификации на несколько частей (этапов), заставляя атакующего применять несколько схем или устройств, чтобы получить обе части ключа и желаемый доступ.

Выявление вторжений

Системы выявления вторжений подробно рассматривались в главе 3. Системы выявления вторжений располагаются позади «передовых рубежей» управления доступом и предназначены для выявления попыток или успешных проникновений в сеть с помощью пассивных наблюдений. Системы предотвращения вторжений (intrusion prevention) позиционируются как усовершенствование систем пассивного выявления вторжений, предоставляющее возможность перехвата прямой линии связи между источником и приемником и автоматического выявления аномалий. Сниффинг (перехват и анализ) пакетов (packet sniffing) в реальном времени является обязательным требованием к любой системе выявления или предотвращения вторжений, поскольку предоставляет полный доступ к контенту, проходящему через периметр сети, и к данным, по которым можно определить угрозу или факт вторжения.

Обнаружение атакующих внутри сети

Если предполагать, что атакующие могут обойти механизмы управления доступом и избежать обнаружения системами выявления вторжений, то следует принять как факт, что они пересекают границу сети и действуют в доверительной среде вашей инфраструктуры. В правильно спроектированных системах должна предусматриваться постоянная готовность к работе по выявлению атакующих внутри сети. Вне зависимости от того, внедрились ли атакующие извне или это нечестные сотрудники организации, системным администраторам необходимо активное и даже агрессивное инструментирование инфраструктуры с помощью средств наблюдения и фиксации в журналах сетевых действий, чтобы расширить область видимости на серверах и между ними. Одной защиты по периметру недостаточно, потому что атакующий, который затратил достаточный объем времени и ресурсов на преодоление периметра, чаще всего добивается успеха.

Правильная сегментация сети может ограничить размер ущерба, причиняемого атакующими, проникшими внутрь сети. С помощью отделения открытых во внешнюю среду систем от внутренних серверов, хранящих важную приватную информацию, позволяя только надежно защищенным и контролируемым API получать доступ в отдельные сегменты сети, администраторы ограничивают каналы, доступные атакующим изнутри, и обеспечивают себе более широкий обзор и возможность тщательного исследования трафика между узлами. Микросегментация

(microsegmentation) – это практическая методика разделения сети на различные секции (сегменты) на основе логической функциональности каждого элемента. Правильная микросегментация может упростить саму структуру сети и управление стратегиями защиты, но ее непрерывная эффективность зависит от наличия четко определенных процессов изменения инфраструктуры. Изменения в сети должны быть точно отражены в схемах сегментации, управление которыми может становиться весьма трудной задачей при постоянно возрастающей сложности систем. В любом случае, сегментация сети дает администраторам возможность получить строгий контроль и управление многочисленными возможными маршрутами между узлом A и узлом B в сети, а также обеспечивает расширенную область видимости, позволяющую воспользоваться методами науки о данных для выявления атак.

Защита, основанная на обработке данных

После того как атакующий преодолел внешний периметр, все данные, хранящиеся в сети, подвергаются риску быть похищенными. Злоумышленники, получившие доступ к сегменту сети, содержащему приватные данные о пользователях, получают также неограниченный доступ ко всей информации, хранящейся в этой сети. В этом случае единственным способом хоть как-то ограничить причиняемый ущерб является использование методики защиты, основанной на обработке данных (data-centric security). В методике защиты, основанной на обработке данных, главное внимание уделяется защите самих данных, т. е. даже после взлома базы данных сами данные могут не иметь большой ценности для взломщика. Зашифрование хранимых данных – один из методов такой защиты. Часто применяемой стандартной практической методикой является зашифрование с использованием специального модификатора (salt) и хеширование хранимых паролей, это реализовано, например, в Unix-подобных операционных системах. Атакующий не может с легкостью воспользоваться похищенными регистрационными данными для доступа к учетной записи пользователя. Но зашифрование хранимых данных может быть применено не во всех средах и контекстах. Для наборов данных, которые интенсивно используются для анализа и/или должны весьма часто предоставляться в незашифрованной форме, постоянное зашифрование и расшифрование может привести к необоснованно высоким требованиям к потреблению ресурсов.

Анализ зашифрованных данных – это целевая задача, над решением которой сообщества по исследованиям в областях обеспечения безопасности и интеллектуального анализа данных (data mining) работают уже в течение длительного времени, а иногда ее решение даже называют своеобразным «Святым Граалем» криптографии[1]. Гомоморфное шифрование (homomorphic encryption) позиционируется как самая многообещающая методика, позволяющая найти окончательное решение этой задачи. Можно воспользоваться схемой полного гомоморфного шифрования, такой как схема Бракерски–Джентри–Вайкунтанатана (Brakerski–Gentry–Vaikuntanathan – BGV)[2] для выполнения вычислительного анализа без

[1] *David Wu*. Fully Homomorphic Encryption: Cryptography's Holy Grail. XRDS: Crossroads, he ACM Magazine for Students 21:3 (2015): 24–29.

[2] *Zvika Brakerski, Craig Gentry and Vinod Vaikuntanathan*. Fully Homomorphic Encryption Without Bootstrapping. Proceedings of the 3rd Innovations in heoretical Computer Science Conference (2012): 309–325.

предварительного расшифрования данных. Это позволяет различным сервисам обработки данных, работающим с одним и тем же фрагментом данных, передавать зашифрованную версию данных друг другу, исключая необходимость расшифровки данных. Даже несмотря на то что схемы гомоморфного шифрования еще не получили широкого распространения, улучшение их производительности и эффективности представляет собой активную область исследований. HElib (**https://github.com/shaih/helib**) – библиотека эффективной реализации гомоморфного шифрования с использованием функций низкого уровня и схемы шифрования BGV.

Приманка для злоумышленников

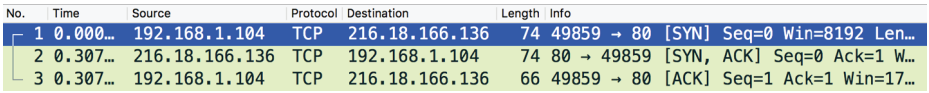
Приманка для злоумышленников (honeypot) – это ловушка, специально предназначенная для сбора информации об атакующих. Существует множество способов создания и настройки серверов-ловушек и сетей-ловушек (последние часто называют honeynets), но главной целью этих средств остается исследование методологий атак и сбор значимой для юридических (судебных) органов информации с целью анализа действий атакующих. Приманки (ловушки) предоставляют интерфейсы, имитирующие реальные системы, и иногда могут достаточно успешно обманывать атакующих, заставляя их раскрывать секретные характеристики атак, которые могут помочь при офлайновом анализе данных и/или при создании активных контрмер. Ловушки стратегически грамотно размещаются в средах, в которых накоплен значительный опыт выявления и отражения атак, что может оказаться полезным для сбора маркированных тренировочных данных для исследования и усовершенствования моделей выявления атак.

Резюме

В предыдущей части этой главы был представлен самый обобщенный обзор обеспечения сетевой безопасности с небольшими подробностями, соответствующими контексту использования машинного обучения для защиты сетей. Это область активных исследований, направленных исключительно на разработку методологий обеспечения безопасности и защиты сетевых сред. Поэтому невозможно полностью рассмотреть все сложности и проблемы в этой области всего лишь в нескольких разделах. В следующей части данной главы мы подробно рассмотрим более конкретные атаки и методы эффективного извлечения полезного опыта обеспечения безопасности из сетевых данных с использованием науки о данных и машинного обучения.

Машинное обучение и обеспечение безопасности сети

Поиск и определение шаблонов является одним из главных достоинств машинного обучения, а в сетевом трафике существует множество характерных шаблонов данных, которые необходимо выявить. На первый взгляд может показаться, что данные в перехваченном сетевом пакете нерегулярны и случайны, но в большинстве потоков обмена информацией используются строго определенные сетевые протоколы. Например, при исследовании перехваченных сетевых пакетов можно наблюдать трехэтапное согласование по протоколу TCP (TCP three-way handshake), как показано на рис. 5.2.



No.	Time	Source	Protocol	Destination	Length	Info
1	0.000…	192.168.1.104	TCP	216.18.166.136	74	49859 → 80 [SYN] Seq=0 Win=8192 Len…
2	0.307…	216.18.166.136	TCP	192.168.1.104	74	80 → 49859 [SYN, ACK] Seq=0 Ack=1 W…
3	0.307…	192.168.1.104	TCP	216.18.166.136	66	49859 → 80 [ACK] Seq=1 Ack=1 Win=17…

Рис. 5.2 ❖ Трехэтапное согласование по протоколу TCP
(Источник: снимок экрана утилиты Wireshark)

После идентификации процедуры согласования, которая обозначает начало соединения по протоколу TCP, можно выделить остальную часть TCP-сеанса. Идентификация TCP-сеансов не является особо сложной задачей, но она позволяет понять, что сетевой трафик строго управляется набором протоколов, зная которые, в результате можно получить структуры и шаблоны данных. Также можно обнаружить вредоносные действия в сетях с помощью определения шаблонов и установления взаимосвязей в данных, особенно при атаках, основанных на большом объеме и/или многократном повторении сетевых операций, как, например, при сканировании сети и DoS-атаках (Denial of Service attacks).

От перехваченных данных к признакам

Оперативный перехват передаваемых по сети данных – это главный способ фиксации сетевой активности для онлайнового и офлайнового анализа. Подобно видеокамере наблюдения за дорожным трафиком на перекрестке, анализаторы пакетов (packet analyzers) (также известные под названием снифферы (sniffers) пакетов/сетей/протоколов) перехватывают и регистрируют в журналах трафик в сети. Сетевые журналы регистрации (logs) полезны не только для исследования инцидентов, связанных с безопасностью, но также для отладки, наблюдения за производительностью и мониторинга сетевых операций. При размещении в надлежащих пунктах сетей[1] и при правильно сконфигурированных сетевых коммутаторах и интерфейсах анализатор пакетов может стать полезнейшим инструментом для генерации подробных наборов данных, дающих полную картину всего происходящего в сети. Разумеется, объем этих данных может быть ошеломляющим. В сложных сетевых средах даже кажущиеся простыми задачи, такие как трассировка TCP-сеанса, которая была показана выше, в действительности совсем не просты. После получения доступа к изобилующим информацией исходным необработанным данным следующим шагом является генерация полезных признаков для анализа данных.

Автоматическое обучение признакам

Не все методики майнинга данных и машинного обучения требуют ручного конструирования признаков. Алгоритмы обучения признакам без учителя (unsupervised feature learning) и глубокого обучения могут автоматически обучаться представлениям признаков и по данным с метками, и по данным без меток, позволяя специалистам-практикам избежать излишних трудозатрат на конструирование и выбор признаков. Следует особо отметить, что «обучение признакам без учителя» отлича-

[1] Анализаторы пакетов существуют как в виде аппаратных устройств, так и в виде программных средств и могут перехватывать либо только заголовки пакетов, либо все содержимое пакетов полностью.

ется от «обучения без учителя» – первый термин обозначает автоматическую генерацию признаков из исходных необработанных данных, тогда как второй термин обозначает машинное обучение с использованием данных без меток. В любом приложении машинного обучения для анализа данных важно рассматривать и сравнивать результаты методик, требующих конструирования признаков, с методиками, использующими обучение признакам без учителя. Ответ на вопрос, какие модели и алгоритмы более эффективны для того или иного набора данных, неоднозначен и в высшей степени зависит от природы самих данных.

Теперь рассмотрим некоторые специализированные методы перехвата данных и генерации признаков из сетевого трафика.

Tcpdump (**http://www.tcpdump.org/**) – это анализатор сетевых пакетов, работающий в командной строке и имеющийся во всех современных Unix-подобных операционных системах. Перехват данных выполняется в файлы формата libpcap (*.pcap*), который является достаточно универсальным и переносимым форматом для перехваченных сетевых данных. Ниже приведен пример использования анализатора tcpdump для перехвата трех TCP-пакетов на всех сетевых интерфейсах:

```
# tcpdump -i any -c 3 tcp

3 packets captured
3 packets received by filter
0 packets dropped by kernel

12:58:03.231757 IP (tos 0x0, ttl 64, id 49793, offset 0,
        flags [DF], proto TCP (6), length 436)
    192.168.0.112.60071 > 93.184.216.34.http: Flags [P.],
cksum 0x184a (correct), seq 1:385, ack 1, win 4117,
options [nop,nop,TS val 519806276 ecr 1306086754],
length 384: HTTP, length: 384
    GET / HTTP/1.1
    Host: www.example.com
    Connection: keep-alive
    Upgrade-Insecure-Requests: 1
    User-Agent: Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_3)
        AppleWebKit/537.36 (KHTML, like Gecko)
        Chrome/56.0.2924.87 Safari/537.36
    Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,
        */*;q=0.8
    Accept-Encoding: gzip, deflate, sdch
    Accept-Language: en-US,en;q=0.8

12:58:03.296316 IP (tos 0x0, ttl 49, id 54207, offset 0, flags [DF],
        proto TCP (6), length 52)
    93.184.216.34.http > 192.168.0.112.60071: Flags [.],
cksum 0x8aa4 (correct), seq 1, ack 385, win 285,
options [nop,nop,TS val 1306086770 ecr 519806276], length 0

12:58:03.300785 IP (tos 0x0, ttl 49, id 54208, offset 0, flags [DF],
        proto TCP (6), length 1009)
    93.184.216.34.http > 192.168.0.112.60071: Flags [P.],
cksum 0xdf99 (correct), seq 1:958, ack 385, win 285,
options [nop,nop,TS val 1306086770 ecr 519806276],
```

```
length 957: HTTP, length: 957
    HTTP/1.1 200 OK
    Content-Encoding: gzip
    Accept-Ranges: bytes
    Cache-Control: max-age=604800
    Content-Type: text/html
    Date: Sat, 04 Mar 2017 20:58:03 GMT
    Etag: "359670651+ident"
    Expires: Sat, 11 Mar 2017 20:58:03 GMT
    Last-Modified: Fri, 09 Aug 2013 23:54:35 GMT
    Server: ECS (fty/2FA4)
    Vary: Accept-Encoding
    X-Cache: HIT
    Content-Length: 606
```

Эти три пакета были переданы между домашним/частным IP-адресом 192.168.0.112 и удаленным HTTP-сервером с IP-адресом 93.184.216.34. Первый пакет содержит запрос GET к HTTP-серверу, второй пакет – это подтверждение получения пакета запроса HTTP-сервером, третий пакет – ответ HTTP-сервера. Хотя tcpdump является мощным инструментом, позволяющим перехватывать, анализировать, фильтровать, выполнять дешифрование и поиск по сетевым пакетам, Wireshark (**https://www.wireshark.org/**) представляет собой достойную альтернативу, которая предоставляет графический интерфейс пользователя и некоторые дополнительные функциональные возможности. Этот анализатор пакетов поддерживает стандартный формат файлов libpcap, но по умолчанию перехватывает пакеты в усовершенствованном формате PCAP Next Generation (.pcapng) (**https://wiki.wireshark.org/Development/PcapNg**).

Transport Layer Security (TLS) (https://tools.ietf.org/html/rfc5246), часто называемый именем своего предшественника Secure Sockets Layer (SSL), – это протокол, обеспечивающий целостность и защиту данных, передаваемых между двумя связанными по сети приложениями. Инкапсуляция данных по протоколу TLS является хорошим средством для обеспечения сетевой безопасности, поскольку неавторизованный сниффер пакетов, размещенный в сетевом маршруте между двумя приложениями, может получить только зашифрованные пакеты, из которых нельзя извлечь полезную информацию. Сетевой аналитик, пытающийся на законных основаниях извлечь информацию из зашифрованного трафика по протоколу TLS, должен выполнить дополнительную операцию по расшифрованию пакетов. Администраторы могут расшифровывать трафик TLS/SSL (часто обозначаемый термином «закрытый SSL» (terminating SSL)), если имеют доступ к секретному ключу, используемому для зашифровки данных, а перехваченные данные включают начальный этап установления сеанса TLS/SSL в соответствии с протоколом согласования TLS Handshake Protocol (**https://tools.ietf.org/html/rfc5246#section-7.3**), где среди прочих параметров представлены секретные характеристики сеанса. В большинстве современных анализаторов пакетов имеется возможность расшифрования пакетов TLS/SSL. Tcpdump не предоставляет такую функцию, но она имеется в утилите ssldump (**http://ssldump.sourceforge.net/**). Wireshark также предлагает очень простой интерфейс, позволяющий выполнять автоматические преобразования (расшифрование) всех зашифрованных пакетов (**https://wiki.wireshark.org/SSL**), если у вас есть секретный ключ и схема зашифрова-

ния. Рассмотрим пример расшифровки содержимого TCP-пакета, перехваченного в сети с функцией зашифрования по протоколу TLS/SSL HTTPS-трафика[1] с использованием схемы шифрования RSA[2]:

```
dissect_ssl enter frame #19 (first time)
packet_from_server: is from server - TRUE
  conversation = 0x1189c06c0, ssl_session = 0x1189c1090
  record: offset = 0, reported_length_remaining = 5690
dissect_ssl3_record: content_type 23 Application Data
decrypt_ssl3_record: app_data len 360, ssl state 0x23F
packet_from_server: is from server - TRUE
decrypt_ssl3_record: using server decoder
ssl_decrypt_record ciphertext len 360

Ciphertext[360]:
dK.-&..R....yn....,.=....,.I2R.-...K..M.!G..<..ZT..]"..V_....'.;..e.
c'^T....A.@pz......MLBH.?.:\.)..C...z5v..........F.w..]A...n........
.w.@. %....I..gy.........c.pf...W.....Xt..?Q.....J.Iix!..O.XZ.G.i/..I
k.*`...z.C.t..|.....$...EX6g.8.......U.."...o.t...9{..X{ZS..NF.....w
t..T.[|[...9{g.;@.!.2..B[.{..j.....;i.w..fE...;.......d..F....4.....
W.....+xhp....p..(-L

Plaintext[360]:
HTTP/1.1 200 OK..Date: Mon, 24 Apr 2006 09:04:18 GMT..Server: Apache
/2.0.55 (Debian) mod_ssl/2.0.55 OpenSSL/0.9.8a..Last-Modified: Mon,
27 Mar 2006 12:39:09 GMT..ETag: "14ec6-14ae-42cf5540"..Accept-Ranges
: bytes..Content-Length: 5294..Keep-Alive: timeout=15, max=100..Conn
ection: Keep-Alive..Content-Type: text/html; charset=UTF-8.....&..FS
...k.r8.Z#[.TfC.....

ssl_decrypt_record found padding 5 final len 354
checking mac (len 334, version 300, ct 23 seq 1)
ssl_decrypt_record: mac ok
```

Без секретного ключа RSA единственное, что может видеть сниффер пакетов, – это зашифрованный текст, который не предоставляет никакой информации о реальном содержимом сообщения, т. е. об ответе HTTP 200 OK.

Далее рассмотрим пример, показывающий, как можно извлечь общие признаки из необработанного содержимого перехваченного сетевого пакета. Предполагается, что читатель имеет некоторое представление о процессе генерации признаков (более подробно об этом *см.* главу 4). В приведенном ниже примере рассматривается имитация TCP-сеанса атакующего, который удаленно использует уязвимости системы, как показано на рис. 5.3.

Можно сразу же получить следующую основную информацию о сеансе:

- продолжительность сеанса: 4.971 с;
- общее количество пакетов в сеансе: 10;
- протокол: TCP;
- общее количество байтов, переданных от источника к приемнику:

---

[1] См. snakeoil2_070531.tgz на **https://wiki.wireshark.org/SampleCaptures#Sample_Captures**.

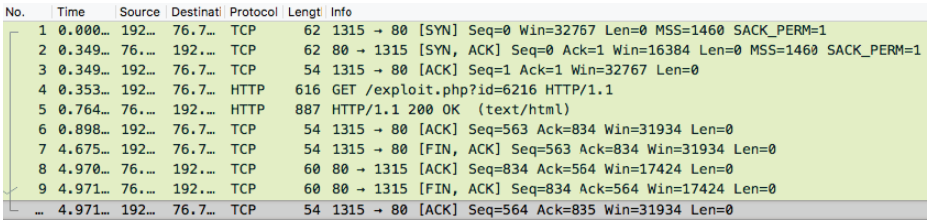
[2] *R. L. Rivest, A. Shamir and L. Adleman*. A Method for Obtaining Digital Signatures and Public-Key Cryptosystems. Communications of the ACM 21 (1978): 120–126.

62 + 54 + 616 + 54 + 54 + 54 = 894 байта;

- общее количество байтов, переданных от приемника к источнику:

  62 + 887 + 60 + 60 = 1069 байтов;

- успешная процедура согласования по протоколу TCP: true (истина);
- сетевой сервис на целевом хосте: HTTP;
- количество пакетов подтверждения ACK: 4.

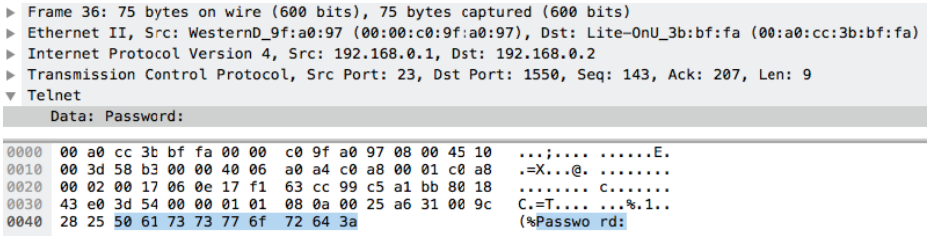


No.	Time	Source	Destinati	Protocol	Lengt	Info
1	0.000…	192…	76.7…	TCP	62	1315 → 80 [SYN] Seq=0 Win=32767 Len=0 MSS=1460 SACK_PERM=1
2	0.349…	76.…	192.…	TCP	62	80 → 1315 [SYN, ACK] Seq=0 Ack=1 Win=16384 Len=0 MSS=1460 SACK_PERM=1
3	0.349…	192…	76.7…	TCP	54	1315 → 80 [ACK] Seq=1 Ack=1 Win=32767 Len=0
4	0.353…	192…	76.7…	HTTP	616	GET /exploit.php?id=6216 HTTP/1.1
5	0.764…	76.…	192.…	HTTP	887	HTTP/1.1 200 OK (text/html)
6	0.898…	192…	76.7…	TCP	54	1315 → 80 [ACK] Seq=563 Ack=834 Win=31934 Len=0
7	4.675…	192…	76.7…	TCP	54	1315 → 80 [FIN, ACK] Seq=563 Ack=834 Win=31934 Len=0
8	4.970…	76.…	192.…	TCP	60	80 → 1315 [ACK] Seq=834 Ack=564 Win=17424 Len=0
9	4.971…	76.…	192.…	TCP	60	80 → 1315 [FIN, ACK] Seq=834 Ack=564 Win=17424 Len=0
…	4.971…	192…	76.7…	TCP	54	1315 → 80 [ACK] Seq=564 Ack=835 Win=31934 Len=0

Рис. 5.3 ❖ TCP-сеанс атакующего взломщика
(Источник: снимок экрана утилиты Wireshark)

Сбор признаков в длинной последовательности пакетов может позволить сгенерировать более полезную информацию из полученных данных, чем анализ отдельных пакетов. Например, при попытках обнаружить SQL-инъекции через сетевые соединения с выполнением анализа на уровне пакетов приходится просматривать огромное количество «ненужных» пакетов из-за издержек протокола обмена данными. С другой стороны, попытка определить лавинную маршрутизацию средствами протокола Internet Control Message Protocol (ICMP) с целью проведения DoS-атаки требует анализа на уровне пакетов, потому что в этом случае сеансов как таковых нет.

Другие признаки, характерные для конкретного приложения, также могут быть извлечены из перехваченных сетевых пакетов. Например, можно извлечь сообщения, передаваемые по протоколу Telnet (**https://tools.ietf.org/html/rfc854**) при запуске процесса-демона, как показано на рис. 5.4.



```
▶ Frame 36: 75 bytes on wire (600 bits), 75 bytes captured (600 bits)
▶ Ethernet II, Src: WesternD_9f:a0:97 (00:00:c0:9f:a0:97), Dst: Lite-OnU_3b:bf:fa (00:a0:cc:3b:bf:fa)
▶ Internet Protocol Version 4, Src: 192.168.0.1, Dst: 192.168.0.2
▶ Transmission Control Protocol, Src Port: 23, Dst Port: 1550, Seq: 143, Ack: 207, Len: 9
▼ Telnet
    Data: Password:

0000  00 a0 cc 3b bf fa 00 00  c0 9f a0 97 08 00 45 10   ...;.... ......E.
0010  00 3d 58 b3 00 00 40 06  a0 a4 c0 a8 00 01 c0 a8   .=X...@. ........
0020  00 02 00 17 06 0e 17 f1  63 cc 99 c5 a1 bb 80 18   ........ c.......
0030  43 e0 3d 54 00 00 01 01  08 0a 00 25 a6 31 00 9c   C.=T.... ...%.1..
0040  28 25 50 61 73 73 77 6f  72 64 3a                 (%Passwo rd:
```

Рис. 5.4 ❖ Секция данных в пакете Telnet
(Источник: снимок экрана утилиты Wireshark)

В этом пакете данных Telnet можно видеть запрос (промпт) пароля. Telnet – это протокол, предназначенный для двунаправленного взаимодействия между двумя сетевыми хостами через виртуальное терминальное соединение. Все данные по протоколу Telnet передаются по сети в обычном виде, что связано с огромным

риском для безопасности. На этапе зарождения и формирования компьютерных сетей это не являлось большой проблемой. Но со временем задача защиты сетей прибретала все большее значение, в конце концов протокол SSH полностью заменил Telnet. В SSH реализованы протоколы надежного шифрования, защищающие обмен данными между хостами от шпионажа или перехвата. Тем не менее Telnet вместе с другими протоколами обмена данными между хостами без шифрования продолжают использовать в частных закрытых сетях, где предполагается, что обеспечение защиты не так важно[1]. Признаки уровня приложения, извлекаемые из данных хорошо известных протоколов, таких как Telnet, могут быть весьма значимыми для анализа данных из перехватываемых сетевых пакетов. Даже в случае зашифрованных каналов обмена информацией может оказаться полезным расшифрование пакетов для извлечения признаков (используя практически те же методы, что и для закрытого TLS/SSL-соединения, описанные выше). Ниже перечислены примеры некоторых признаков, которые можно извлечь:

- протокол прикладного уровня (Telnet, HTTP, FTP или SMTP);
- признак зашифрованных данных;
- неудачная попытка регистрации (входа – login);
- успешная попытка регистрации (входа);
- попытка получения прав доступа суперпользователя root (например, выполнение команды `su root`);
- факт получения прав доступа суперпользователя root;
- признак гостевой учетной записи;
- попытка выполнения команд curl/wget;
- выполненная операция создания файла.

Как было отмечено в предыдущих главах, извлекаемые непрерывные или дискретные значения признаков могут быть представлены в векторах признаков для наиболее удобного использования в алгоритмах анализа данных и машинного обучения с целью получения необходимой информации.

Угрозы в сетевой среде

Прежде чем начать подробное рассмотрение примера анализа сетевых данных, важно обсудить модель потенциальных угроз (опасностей). Как отмечалось выше, мы будем анализировать атаки, относящиеся к сетевому, транспортному и сеансовому уровням (в модели OSI это уровни 3, 4 и 5 соответственно). Несмотря на то что атаки на физическом уровне (уровень 1 [L1] OSI) и на канальном уровне (уровень 2 [L2] OSI) не менее важны при обсуждении обеспечения безопасности сети в целом, здесь мы не рассматриваем их, поскольку варианты реализации и защиты на уровнях L1 и L2 не относятся к компетенции разработчиков приложений и инженеров по обеспечению безопасности. Поэтому практическая реализация средств защиты на этих уровнях обычно не выполнима, если только вы не являетесь специалистом по проектированию и разработке сетевых физических устройств или системного сетевого ПО.

---

[1] В действительности в дистрибутивы многих операционных систем продолжают включать сервис Telnet, устанавливаемый и активируемый по умолчанию. Администраторы, которые намерены обеспечить защиту своих систем от Telnet-атак, должны явно запретить этот сервис или тщательно фильтровать Telnet-трафик.

Потенциальные угрозы (опасности) мы в широком смысле разделим на две категории: пассивные атаки (passive attacks) и активные атаки (active attacks) – с дальнейшим подразделением активных атак на четыре класса: взлом и вторжение (breaches), спуфинг, или обманная имитация (spoofing), закрепление опорного пункта во внутренней сетевой среде (pivoting) с последующим «горизонтальным перемещением» (lateral movement) и отказ в обслуживании (denial of service – DoS).

Пассивные атаки

Пассивные сетевые атаки не инициализируют обмен данными с узлами атакуемой сети, не взаимодействуют с сетевыми данными и не изменяют их. Обычно атакующие применяют методы пассивных атак для сбора информации и операций по рекогносцировке. Сканирование портов (port scanning) – одна из пассивных сетевых атак, применяемая злоумышленниками для обнаружения открытых портов и определения сервисов, работающих на серверах. Обладая знаниями о портах, назначаемых по умолчанию для конкретных сервисов и приложений, злоумышленник может узнать тип (специализацию) сервера только на основе открытых на нем портов, например по открытому порту 27017 можно предположить, что на сервере работает экземпляр СУБД MongoDB. Прослушивание и перехват данных в интернете (Internet wiretapping) (описанный ниже в примечании) представляет собой одну из форм пассивной атаки, которая относится к физическому уровню.

Атаки на физическом уровне

Атаки на физическом уровне, т. е. на уровне физических носителей 802.3 Ethernet, 802.11 Wi-Fi и Bluetooth, нередко обсуждаются в публикациях на тему обеспечения безопасности сетей, поскольку встречаются весьма часто. Помимо лишения работоспособности физических сетевых устройств или подключения электронных снифферов к сегментам сетей, в этой категории важно также учитывать атаки на беспроводные механизмы передачи данных, такие как Wi-Fi и Bluetooth. Aircrack-ng (**http://www.aircrack-ng.org/**) является примером программы для взлома (cracking) Wi-Fi, которая автоматизирует эксплуатацию конкретных слабых мест в специализированных протоколах, защищающих сети Wi-Fi. Лавинная маршрутизация по MAC-адресам (MAC flooding) – это пример атаки на канальном уровне с переполнением таблицы MAC-адресов в сетевых коммутаторах с целью замены корректных записей на некорректные, вследствие чего сетевые пакеты перенаправляются в совершенно неожиданные сегменты сети.

Прослушивание и перехват данных в интернете (Internet wiretapping) – еще одна форма пассивной сетевой атаки на физическом уровне. При такой атаке выполняется перехват сетевого трафика в некотором (промежуточном) пункте передачи, позволяющий взломщику получать данные даже без знаний о сетевых пользователях. Атаки посредника типа «человек посередине» (man-in-the-middle) часто используются для практической реализации прослушивания и перехвата данных в интернете и позволяют атакующим просматривать приватный (секретный) трафик между хостами и/или между пользователями-людьми.

Активные атаки

Активные атаки гораздо более разнообразны и могут быть классифицированы по следующим категориям:

- взлом и вторжение: вторжение через уязвимые места в сети (network breaches) – возможно, самый часто встречающийся тип сетевых атак. Термин «уязвимое место» (breach) может обозначать «пробоину», слабый участок периметра внутренней сети или действие атакующего, использующего такую пробоину для получения неавторизованного доступа к приватным системам. Во многих серверных инфраструктурах сетевые узлы расположены на периметре. Примерами таких узлов могут служить маршрутизаторы, прокси-серверы, сетевые защитные экраны, коммутаторы и балансировщики сетевой нагрузки. Системы выявления вторжений – одна из форм защиты периметра. Такие системы пытаются обнаружить атакующего в момент, когда он использует уязвимые места периметра для получения доступа в сеть. Как отмечалось в главе 3, системы выявления вторжений являются часто применяемым вариантом практического применения методики выявления аномалий.
  Атакующие могут выполнять вторжение в сети после пассивного сбора информации и рекогносцировки (разведки) с целью поиска уязвимых мест в общедоступных конечных пунктах, в которых можно получить доступ к командной оболочке или права суперпользователя root в системах. Команды, выполняемые по сети в попытках использовать уязвимости приложений, могут быть обнаружены с помощью наблюдения за каналами обмена данными между серверами, именно поэтому атаки на приложения с удаленных узлов иногда классифицируют как взлом и вторжение в сеть (network breach). Например, атаки с целью переполнения буферов или динамически распределяемой памяти, SQL-инъекции и межсайтовый скриптинг (XSS), по существу, не являются следствием проблем в сети, но при постоянном наблюдении за трафиком между хостами системы сетевой защиты способны обнаружить подобные попытки взлома серверов внутри сети. Например, попытка вызвать простое переполнение буфера памяти с удаленного узла может быть обнаружена при проверке содержимого сетевого пакета на наличие в нем конкретных сигнатур атаки и заблокирована, или же такой пакет помещается в карантин. Тем не менее многообразие форм таких удаленно инициируемых атак приводит к тому, что методика проверки опасных сигнатур становится практически бесполезной. Это область, в которой применение машинного обучения для обнаружения неточных совпадений может помочь изменить ситуацию.
  Взлом данных инсайдерами (т. е. людьми, работающими непосредственно в организации) также представляет собой значительную сетевую угрозу. Выявление инсайдерских угроз (insider threat detection) ориентировано на обнаружение ситуаций, в которых люди-пользователи, обладающие правами на совершение некоторых действий в системе, создают угрозу этой системе (компрометируют систему) (например, подкупленные сотрудники пытаются продать секретную бизнес-информацию конкурентам). Инсайдерские угрозы являются особенно сложной проблемой, так как системные администраторы обычно имеют неограниченные права доступа ко всей инфраструктуре, а кроме того, могут управлять системами защиты, предотвращающими попытки незаконного доступа к секретным данным. Поверхность атаки можно уменьшить, создав правильные стратегии управления

доступом на основе ролей и систему проверок и регулирования управления и аудита внутренних систем защиты, а также систему шифрования хранимых данных. Решая эту задачу с помощью науки о данных, можно также рассматривать выявление инсайдерских угроз как задачу классификации или выявления аномалий с поиском нарушений целостности и несогласованностей в шаблонах доступа для обнаружения случаев компрометации надежных до сих пор пользователей;

- спуфинг (spoofing): атакующие используют спуфинг (т. е. передачу фальсифицированных данных) как механизм своего внедрения и закрепления в середине надежного канала обмена информацией между двумя объектами. DNS-спуфинг и ARP-спуфинг (также известные под названием «отравление кеша» (cache poisoning)) используют в своих корыстных целях механизмы сетевого кеширования, чтобы заставить клиента вести обмен информацией с подставным ложным объектом, а не с объектом, который изначально имел в виду клиент. Выдавая себя за реальную принимающую сторону, атакующие в дальнейшем могут предпринимать пассивные атаки прослушивания для похищения информации из считавшихся до этого конфиденциальными каналов обмена данными.
  DNS-серверы выполняют преобразование имен доменов, удобных для чтения человеком (например, *www.example.com*), в цифровые IP-адреса по правилам протокола DNS. Атакующий может «отравить», т. е. изменить DNS-кеш клиента, чтобы временно перенаправить его на нелегальный DNS-сервер, который будет отвечать на запросы DNS-преобразования, передавая IP-адрес злоумышленника. Набор расширений DNSSEC (**https://www.ietf.org/rfc/rfc4033.txt**) был введен для обеспечения аутентичности и целостности процесса DNS-преобразований, что позволяет предотвратить большинство атак DNS-спуфинга;
- закрепление опорного пункта во внутренней сетевой среде (pivoting) – это методика перемещения между серверами во внутренней сети, после того как атакующий получил доступ к точке входа в сеть. Инфраструктуры, в которых границы между сервисами неправильно спроектированы и сконфигурированы, особенно уязвимы для атак этого типа. Многие случаи взлома важных секретных данных осуществлены атакующими, внедрившимися и действующими во внутренней сети после получения доступа к малозначимому хосту. Малозначимые хосты – это хосты в сети, которые не предоставляют атакующим почти никакой информации даже после их взлома. Обычно это системы, открытые для внешней среды, например веб-серверы интернета или общедоступные терминалы оплаты покупок. Эти системы изначально не предназначены для хранения информации, представляющей хоть какую-то ценность для злоумышленников. Поэтому их системы защиты обычно гораздо слабее по сравнению с более важными системами, такими как серверы бизнес-логики или базы данных клиентов.
  В сети с хорошо организованной защитой обмен информацией между малозначимыми хостами и важными внутренними хостами должен быть разрешен только через небольшой и управляемый набор каналов связи. Но многие сети неправильно сегментированы и содержат неконтролируемые зоны (blind spots), которые позволяют атакующим перемещаться из одного

сегмента виртуальной локальной сети (VLAN) в другой и в конечном итоге находить путь к важному внутреннему хосту. Такие методики, как имитация коммутатора (switch spoofing) и дублирование VLAN-тегов (double tagging), позволяют атакующим выполнять последовательное перемещение по виртуальной локальной сети (VLAN hopping) (**https://en.wikipedia.org/wiki/VLAN_hopping**) через неправильно сконфигурированные интерфейсы VLAN. Кроме того, атакующие могут воспользоваться взломанным малозначимым хостом для выполнения атак «грубой силой» (brute-force), фаззинга (fuzzing) или сканирования портов в остальной части сети для поиска очередного опорного пункта как основы для дальнейших перемещений внутри сети. Meterpreter (**https://metasploit.help.rapid7.com/docs/the-payload-generator**) представляет собой инструментальное средство, предназначенное для автоматизации процесса внедрения и закрепления опорного пункта в сети. Этот инструмент можно использовать для тестирования процесса внедрения, чтобы найти уязвимые места в вашей сети;

- DoS-атаки: атаки типа «отказ в обслуживании» (denial-of-service – DoS) нацелены на нарушение общей доступности системы, затрудняя и даже полностью исключая доступ к ней для всех желающих пользователей. Существует множество вариантов DoS-атак, в том числе распределенная (distributed) DDoS-атака, при которой используются многочисленные IP-адреса, иногда расположенные в различных геолокациях.

  SYN-флуд (SYN flooding) – методика DoS-атаки, при которой некорректно применяется трехэтапный механизм TCP-согласования. Атакуются неправильно реализованные конечные сетевые пункты (узлы). В процессе трехэтапного согласования по протоколу TCP каждое новое TCP-соединение начинается с того, что клиент отправляет пакет SYN на сервер. Сервер отвечает клиенту пакетом SYN-ACK и ждет в течение определенного интервала времени ответа с пакетом подтверждения ACK от клиента. Сервер обязательно должен сохранять и поддерживать полуоткрытое соединение в ожидании пакета подтверждения ACK от клиента, предполагая, что задержка может произойти по различным причинам, например из-за неустойчивости сетевых каналов связи или из-за перегрузок (заторов) в сети. Поддержка таких полуоткрытых соединений требует использования системных ресурсов в течение некоторого интервала времени. Клиенты-злоумышленники могут постоянно инициировать TCP-соединения, буквально затопляя сервер пакетами SYN и не отвечая пакетами подтверждения ACK. В конце концов, сервер исчерпывает все доступные ресурсы и отказывает добропорядочным клиентам в инициализации новых соединений[1]. Существует множество других вариантов DoS-атак, которые точно так же приводят к полному исчерпанию ресурсов сервера и недоступности соответствующего сервиса.

  Ботнет (botnet) – это сеть скомпрометированных (взломанных) находящихся в частном владении компьютеров, которые заражены (без ведома владельцев) вредоносным ПО и используются для удаленного доступа. У ботнетов множество применений, в том числе рассылка спама, выполнение

[1] В документе RFC 4987 (**https://tools.ietf.org/html/rfc4987**) дано описание атак типа SYN-флуд и приводятся рекомендуемые контрмеры по защите от атак этого типа.

кликфрода (click fraud)[1] и искажение веб-контента, но чаще всего ботнет применяется для организации DoS-атак. Важность ботнетов настолько велика, что в следующем разделе мы более подробно рассмотрим этот сетевой объект.

Ботнет и защита от него

Почти половина всего веб-трафика генерируется ботами (**https://www.computerworld.com/article/3070058/half-the-webs-traffic-comes-from-bots.html**). Боты существуют во многих формах, иногда как простой скрипт командной оболочки bash, состоящий из команд curl (**https://curl.haxx.se/**), иногда как браузер без графического пользовательского интерфейса, но с поддержкой скриптов, например PhantomJS (**http://phantomjs.org/**), иногда даже в виде крупномасштабного распределенного поискового робота (web crawler), поддерживаемого программной средой, подобной Apache Nutch (**http://nutch.apache.org/**). Такие боты иногда проявляют способности к обучению, неторопливо перемещаясь по веб-сайтам для индексирования поискового механизма под управлением правил, определенных на сайте **http://www.robotstxt.org/** в файле robots.txt. Другие боты не так дружелюбны и могут проявлять даже враждебные намерения. В одном из исследований утверждалось, что источниками 28,9 % от всего трафика в интернете можно признать вредоносные боты, хотя это значение весьма приблизительно и может существенно отличаться от действительной ситуации. Вредоносные боты преднамеренно искажают веб-содержимое, похищают секретные регистрационные данные из форм регистрации и входа в системы, участвуют в DoS-атаках, рассылают спам и фишинговые письма и т. д. Как отмечалось в предыдущих главах, зомби-узлы в ботнетах (т. е. узлы, удаленно управляемые с помощью вредоносного ПО) также считаются ботами, и они почти всегда имеют преступные намерения. Крупные ботнеты могут подчинять себе подключенные к интернету компьютеры по всему миру, позволяя управляющим ими злоумышленникам расширять сферу своей деятельности без соответствующего увеличения накладных эксплуатационных расходов. С учетом того, что распределенные атаки могут исходить от «сервисов надежно защищенного хостинга» (**https://en.wikipedia.org/wiki/Bulletproof_hosting**), размещенных в локациях, в которых давно уже наблюдается повышенный уровень вредоносного трафика, веб-администраторы могут создать простое решение для блокировки таких IP-адресов с плохой репутацией или провайдеров интернета. Тем не менее ботнеты, сформированные из домашних персональных компьютеров, которые могут генерировать большие объемы вполне легального трафика, способны сделать задачу разделения вредоносного и полезного трафика намного более сложной.

Важность понимания сущности ботнетов

Важно обладать достаточным уровнем знаний о ботнетах, поскольку потенциальная угроза, которую они представляют для внутренних сетей организаций и корпоративных сетей, весьма велика. Зомби-узлы, являющиеся частью сети, могут не

[1] Генерация искусственных кликов (щелчков) по рекламным ссылкам, размещенным на некотором веб-сайте с целью получения мошеннической прибыли для хоста этого веб-сайта и/или исчерпания рекламного бюджета рекламодателя.

рассматриваться как активные атакующие сетевые элементы, но при активизации ботнета даже их деятельность может привести к нежелательным последствиям. Как и при APT-атаках и сетевых вторжениях извне, ботнет-зомби способны инициализировать инсайдерские атаки, приводящие к утечкам важной информации, нарушению целостности систем и внесению хаоса в атакуемую сетевую среду. Изучение топологии ботнета и принципов его работы может помочь понять, что именно необходимо искать при попытках определить зараженные хосты в вашей сети.

Несмотря на то что в этой книге мы не рассматриваем специализированные методики определения ботнетов, машинное обучение и статистические методы уже играют важную роль в борьбе против ботов. Можно воспользоваться методикой анализа DNS-запросов[1] или интеллектуальным формированием шаблонов для поиска характеристик сетевого трафика, которые определяют поведение автономного бота. Боты, использующие методику быстрой смены местоположения в сети fast flux networks для маскировки серверов управления и контроля, также могут быть выявлены методами машинного обучения[2]. Кроме того, методы кластеризации применяются к перехваченным данным сетевого трафика для обнаружения каналов обмена информации между зомби и серверами управления и контроля на основе знаний о топологических схемах ботнетов[3].

Как работают ботнеты

Ботнеты – это во всех отношениях распределенные системы. Благодаря высокой материальной заинтересованности в них эти системы часто обладают аккуратной, устойчивой к критическим сбоям архитектурой с высокой степенью доступности, которая способна успешно противостоять любым филантропам, пытающимся прекратить их работу. Зомби-машины в ботнетах, т. е. боты (bots), обычно управляются узлами, делегирующими задачи (серверами управления и контроля). После заражения компьютера вредоносным ПО ботнета сразу же происходит установка некоторого скрытого процесса-демона, который ждет инструкций от сервера управления и контроля. В большинстве случаев этот процесс-демон согласован и логически связан с современными архитектурами оркестровки сервера. Первые ботнеты использовали протоколы Internet Relay Chat (IRC) для обмена информацией с сервером управления и контроля. После внедрения и установки серверный процесс IRC-демона[4] должен быть активизирован и находиться в состоянии ожидания инструкций, передаваемых по предварительно определенно-

---

[1] *Leyla Bilge et al.* EXPOSURE: A Passive DNS Analysis Service to Detect and Report Malicious Domains. ACM Transactions on Information and System Security 16:4 (2014).

[2] *Z. Berkay Celik and Serna Oktug*. Detection of Fast-Flux Networks Using Various DNS Feature Sets. IEEE Symposium on Computers and Communications (2013).

[3] *Guofei Gu et al.* BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection. Proceedings of the 17th USENIX Security symposium (2008): 139–154.
Florian Tegeler et al. BotFinder: Finding Bots in Network Traffic Without Deep Packet Inspection. Proceedings of the 8th International Conference on Emerging Networking Experiments and Technologies (2012): 349–360.

[4] Примеры некоторых IRC-демонов размещены на сайтах **https://www.unrealircd.org** и **http://www.inspircd.org/**.

му каналу связи. После этого администратор ботнета может выполнять различные команды в этих каналах, например следующим образом[1]:

```
> ddos.synflood [host] [time] [delay] [port]
> (execute|e) [path]
> cvar.set spam_maxthreads [8]cvar
> spam.start
```

На раннем этапе существования ботнетов протокол IRC был самым очевидным и простым вариантом выбора из-за своей широчайшей распространенности и легкости развертывания. Кроме того, это была технология, хорошо знакомая создателям ботнетов как пользователям, осуществлявшим крупномасштабную нелегальную деятельность в каналах IRC. Развитие топологий ботнета со временем демонстрирует, как архитектуры могут усовершенствовать собственную устойчивость к критическим сбоям, гибкость и жизнеспособность. В обобщенном смысле существуют четыре категории архитектур управления и контроля (C&C):

- централизованные сети/схема «звезда»: обычная методика управления и контроля ботнетом (рис. 5.5) использует простейшую архитектуру, организованную по следующей схеме: один центральный сервер управления и контроля передает команды для выполнения всем зомби-машинам. Это топология под названием «звезда» (star) обеспечивает непосредственный обмен информацией с зомби-машинами, но очевидна нестабильность такой схемы из-за наличия единой точки отказа (SPoF) в системе. Если сервер управления и контроля выйдет из строя, то администраторы потеряют доступ к зомби. Зомби-машины в этой конфигурации в основном используют DNS как механизм поиска своего сервера управления и контроля, так как адреса серверов управления и контроля должны быть жестко закодированы в ботнет-приложении, а использование DNS-имени вместо IP-адреса позволяет перейти на другой уровень косвенной адресации (т. е. обеспечивает бóльшую гибкость) в такой часто изменяющейся системе.
  Исследователи в области обеспечения безопасности научились выявлять зомби-машины ботнетов в сети при помощи поиска подозрительно выглядящих DNS-запросов[2]. Если обнаружен следующий DNS-запрос, обращенный к локальному DNS-резолверу, то можно вполне обоснованно определить это как проблему:

```
Domain Name System (query)
    Questions: 1
    Answer RRs: 0
    Authority RRs: 0
    Additional RRs: 0
    Queries
        botnet-cnc-server.com: type A, class IN
```

---

[1] Примеры команд Agobot – широко известного IRC-ботнета – описаны в документе The Evolution of Malicious IRC Bots (**https://www.symantec.com/avcenter/reference/the.evolution.of.malicious.irc.botnets.pdf**), автор Джон Кэнэвэн (John Canavan), Symantec Security Response.

[2] *Christian Dietrich et al*. On Botnets That Use DNS for Command and Control. Proceedings of the 7th European Conference on Computer Network Defense (2011): 9–16.

```
        Name: botnet-cnc-server.com
Type: A (Host address)
Class: IN (0x0001)
```

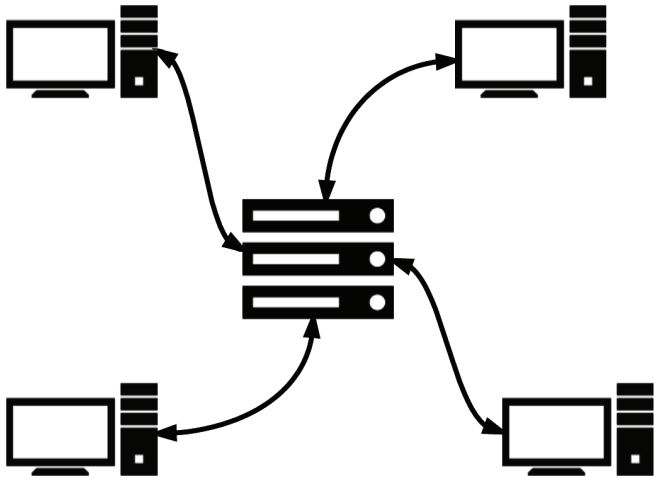


Рис. 5.5 ❖ Схема ботнета с центральным сервером управления и контроля/топология «звезда»

Как и в ситуации с семейством вредоносного ПО, которое немедленно устанавливает связь с центральным сервером и сообщает все локальные характеристики, ботнеты должны применять алгоритмы генерации домена (domain generation algorithms – DGA)[1] для маскировки DNS-запросов, создавая рекурсивный запрос поиска вместо прямого поиска DNS-имени, например *pmdhf98asdfn.com*. Такая маскировка существенно затрудняет точную классификацию запроса как подозрительного. Разумеется, в связи с этим исследователи в области обеспечения безопасности начали разработку эвристических методов[2] и моделей статистического/машинного обучения[3] для выявления подобных искусственно сгенерированных доменных имен, в то время как авторы ботнетов продолжали попытки сделать имена этих доменов выглядящими как можно более безобидно. Это еще один пример бесконечной игры в кошки-мышки;

- сети с мультиуправлением (multileader networks): топология ботнета с мультиуправлением отчасти похожа на схему с центральным сервером управления и контроля, но спроектирована особым образом с целью исключения единой точки отказа. На рис. 5.6 можно видеть, что эта топология значи-

[1] *Phillip Porras, Hassen Saidi and Vinod Yegneswaran*. An Analysis of Conficker's Logic and Rendezvous Points. SRI International Technical Report (2009).

[2] *Sandeep Yadav et al*. Detecting Algorithmically Generated Malicious Domain Names. Proceedings of the 10th ACM SIGCOMM Conference on Internet Measurement (2010): 48–61.

[3] *Hyrum S. Anderson, Jonathan Woodbridge and Bobby Filar*. DeepDGA: Adversarially-Tuned Domain Generation and Detection. Proceedings of the 2016 ACM Workshop on Artiicial Intelligence and Security (2016): 13–21.

тельно повышает уровень сложности сети: серверы управления и контроля обязательно должны часто обмениваться информацией друг с другом, синхронизация становится важнейшей задачей, а общая координация требует больше усилий и трудозатрат. С другой стороны, топология сети с мультиуправлением может немного смягчить проблемы, возникающие из-за физической удаленности узлов сети. Для ботнетов, охватывающих крупные географические регионы и содержащих зомби-машины, которые постоянно обмениваются информацией с серверами управления и контроля почти по всему миру, это становится источником неэффективности и повышает шансы на их обнаружение, поскольку каждый канал обмена информацией потребляет системные ресурсы. Распределение серверов управления и контроля по всему миру как средств доставки сетевого контента с помощью веб-среды является хорошим способом устранения этой проблемы. Тем не менее при этом сохраняется необходимость в определенных типах DNS-сервера или сервера балансировки нагрузки при обмене информацией зомби-машин с кластером управления и контроля. Кроме того, эта топология не решает проблему организации канала связи каждой зомби-машины с центральным командным пунктом;

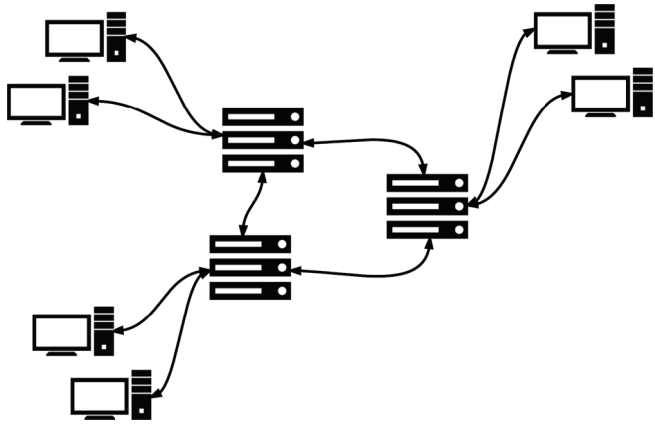


Рис. 5.6 ❖ Схема ботнета с мультиуправлением

- иерархические сети (hierarchical networks): иерархические топологии управления и контроля были созданы специально для устранения проблемы, связанной с единственным командным узлом. Как показано на рис. 5.7, зомби-машины больше не являются только слушателями, теперь они также могут действовать как промежуточные пункты распространения принятых команд. Администраторы ботнета могут передавать команды небольшой группе зомби-машин, подключенных напрямую, которые, в свою очередь, распространяют команды по своим зомби-потомкам и далее по всей сети. Нетрудно представить, насколько трудоемкой становится задача прослеживания источника команд, поэтому такую топологию гораздо сложнее обнаружить, чем схему с центральным сервером управления и контроля (или с несколькими серверами). Но из-за того, что распространение команд по

сети требует некоторого времени, эта топология может оказаться не подходящей для деятельности, которая требует ответов или реакции (действий) в реальном времени. Более того, отказ в работе одной из зомби-машин может сделать недоступным для администраторов ботнета крупный сегмент сети;

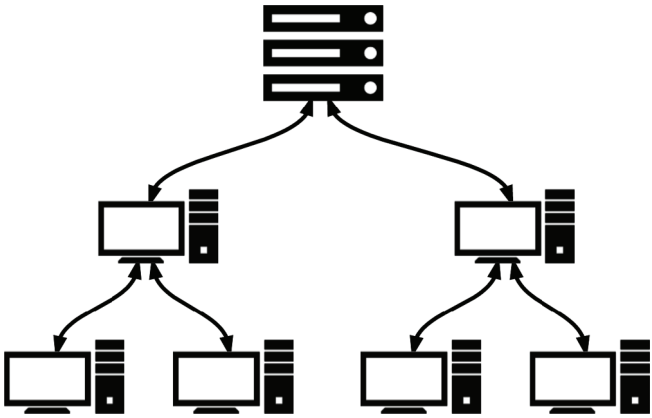


Рис. 5.7 ❖ Иерархическая схема ботнета

- пиринговые (P2P) сети с произвольным доступом: следующим этапом развития топологии ботнета стали полностью децентрализованные системы зомби-машин, напоминающие пиринговую (peer-to-peer – P2P) сеть[1] (см. рис. 5.8). Из этой топологии полностью исключены какие-либо типы централизованных механизмов управления и контроля и доведена до максимума концепция распределенной передачи команд. Администраторы ботнета могут передавать команды любому боту или нескольким ботам в сети, затем эти команды в широковещательном стиле распространяются по всей сети. В результате формируется чрезвычайно гибкая система, поскольку отказ в работе одного бота не оказывает воздействия на другие боты в сети. Но и для этой топологии характерны определенные сложности. Проблема задержки в распространении команд не решена и может вызвать затруднения в обеспечении выполнения и подтверждения команд всеми ботами в сети из-за отсутствия центрального управляющего узла и/или формального протокола управления потоком информации. Кроме того, авторы ботнета обязательно должны спроектировать механизм выполнения команд так, чтобы защитить свои ботнеты от перехвата управления ими третьими сторонами. Поскольку команды управления ботнетом могут выполняться на любом отдельном боте, включая расположенные (возможно) в нежелательных или ненадежных средах выполнения (с точки зрения администраторов ботнета), чрезвычайно важен надежный механизм для обеспечения аутентификации выполняемых команд. Общепринятой методикой достижения этого результата в современных пиринговых ботнетах является

[1] *Ping Wanget al*. A Systematic Study on Peer-to-Peer Botnets. Proceedings of the 18th International Conference on Computer Communications and Networks (2009): 1–8.

создание администраторами подписей выполняемых команд средствами асимметричной криптографии, так что аутентификация команд может выполняться децентрализованным и вполне надежным способом.

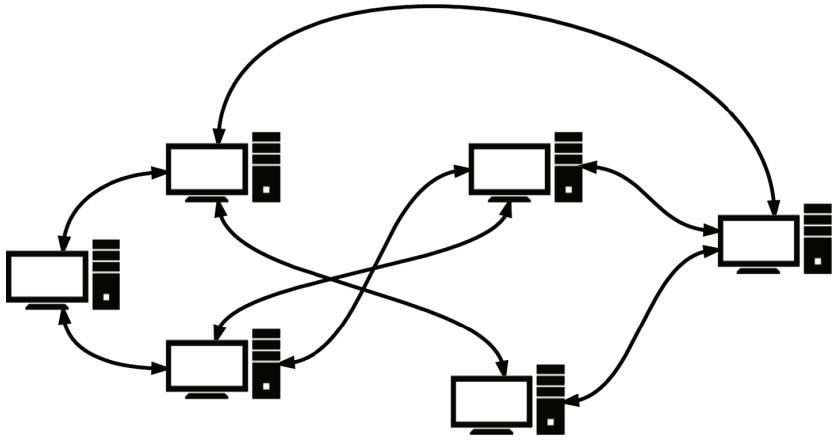


Рис. 5.8 ❖ Схема ботнета как пиринговой сети с произвольным доступом

Создание модели прогнозирования для классификации сетевых атак

В следующей части этой главы будет показано, как создать с нуля классификатор сетевых атак с применением методов машинного обучения. В примере используется набор данных NSL-KDD[1], представляющий собой улучшенную версию классического набора данных для выявления сетевых вторжений, широко применяемый специалистами-профессионалами в области науки о данных, относящейся к обеспечению безопасности[2]. Первоначально набор данных 1999 KDD Cup (**https://kdd.ics.uci.edu/databases/kddcup99/task.html**) был создан для программы исследований DARPA Intrusion Detection Evaluation Program, подготовленной и управляемой научной лабораторией MIT Lincoln Laboratory. Данные были собраны в течение девяти недель и состоят из необработанного трафика, перехваченного утилитой tcpdump в локальной сети (LAN), которая имитировала сетевую среду настоящей локальной сети ВВС США United States Air Force LAN. Некоторые сетевые атаки были преднамеренно совершены в течение этого периода регистрации и фиксации. За это время было зарегистрировано 38 различных типов атак, но в тренировочном наборе доступны только 24 типа. Атаки в тренировочном наборе принадлежат к четырем общим категориям:

- `dos` – отказ в обслуживании;
- `r2l` – неавторизованный доступ с удаленных серверов;

---

[1] *Mahbod Tavallaee et al*. A Detailed Analysis of the KDD CUP 99 Data Set. Proceedings of the 2nd IEEE Symposium on Computational Intelligence in Security and Defense Applications (2009): 53–58.

[2] Этот набор данных можно взять в репозитории кода для данной книги, в разделе *chapter5/datasets* (**https://github.com/oreilly-mlsec/book-resources/tree/master/chapter5/datasets**).

- `u2r` – попытки повышения привилегий;
- `probe` – пробные атаки с помощью «грубой силы».

Этот набор данных в определенной степени является искусственным, поскольку представляет собой набор данных с метками, состоящим из предварительно сгенерированных векторов признаков. В большинстве реальных сценариев гораздо труднее встретить хорошо организованные снабженные метками данные, а конструирование числовых признаков из исходных необработанных данных потребует большей части трудозатрат.

Получение правильных тренировочных данных – постоянно существующая задача, которую необходимо решать при использовании машинного обучения для обеспечения безопасности. Классификаторы настолько эффективны, насколько правильно подобраны данные для их тренировки, а данные с надежными, обоснованными метками особенно важны для машинного обучения с учителем. Большинство организаций вряд ли сможет собрать большой объем трафика атак разнообразных типов, следовательно, необходимо подробно объяснить, как решать задачу с несбалансированными классами. Данные обязательно должны быть размечены с помощью механизма оракула (oracle)[1], который способен точно классифицировать элементы выборки. В некоторых случаях такой способ присваивания меток может быть выполнен операторами без каких-либо специальных навыков, например присваивание меток изображениям животных или чувства и настроения, выражаемые фразами (высказываниями). При классификации сетевого трафика эта задача, как правило, должна выполняться опытными аналитиками в сфере обеспечения безопасности с достаточным уровнем квалификации для научных исследований и судебных расследований. Итоговые результаты работы центров по обеспечению безопасности чаще всего можно преобразовать в некоторую форму тренировочных данных, но этот процесс требует больших затарат времени. В дополнение к использованию методов обучения с учителем для тренировки классификатора также необходимо экспериментировать с применением методов обучения без учителя, т. е. без учета меток, представленных в тренировочном наборе данных.

При отсутствии надежного способа генерации тренировочного набора данных из того же источника, из которого формируются тестовые данные, еще одним альтернативным решением является тренировка классификатора на сравнимом по сущности наборе данных, часто полученном из другого источника или в результате академических научных исследований. Этот метод может успешно применяться в некоторых случаях, но далеко не во всех. Перенос обучения (transfer learning), или индуктивный перенос (inductive transfer), – это процесс принятия модели, тренированной на одном наборе данных, и последующая адаптация (специализация) этой же модели для решения другой родственной задачи. Например, при переносе обучения можно взять общий классификатор изображений[2], который

---

[1] Механизм оракула, или оракул-машина, определен в теории сложности вычислений как некоторая абстрактная сущность, которая способна решать задачи с определенной степенью правильности (точности) за одну операцию.

[2] Классификатор на основе глубокой нейронной сети ImageNet (**http://www.image-net.org/**) успешно применялся во многих приложениях как основа для переноса обучения. Чтобы подробнее узнать о функциональных свойствах ImageNet, которые обеспечивают его успешное применение для этих целей, рекомендуется ознакомиться с документом: *Minyoung Huh, Pulkit Agrawal and Alexei E. Efros*. What Makes ImageNet Good for Transfer Learning. Berkeley Artificial Intelligence Laboratory, UC Berkeley (2016).

способен отличить одежду от сумки, и изменить его, чтобы получить классификатор, способный распознавать различные типы одежды. В настоящее время методика переноса обучения является областью активных научных исследований. Можно привести множество примеров ее успешного применения для классификации текстов[1], фильтрации спама[2] и в байесовских сетях (Bayesian networks)[3].

В предыдущих главах было отмечено, что создание высокоэффективных систем машинного обучения представляет собой процесс с немалой долей исследований и экспериментов. В нескольких следующих разделах подробно рассматривается процесс исследования набора данных и подготовки его для обработки. Затем будут представлены некоторые алгоритмы классификации, которые можно применить для решения поставленной задачи. Суть этого практического примера не в том, чтобы довести читателя до конечного пункта решения задачи, главное – чтобы провести его через все отборочные и квалификационные этапы и помочь сохранить хорошую форму для основного этапа.

Исследование данных

Начнем с более глубокого изучения имеющихся в наличии данных. Тренировочные данные с метками в виде списка значений, разделенных запятыми (формат CSV), выглядят следующим образом:

```
0,tcp,ftp_data,SF,491,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,2,2,0.00,0.00,
0.00,0.00,1.00,0.00,0.00,150,25,0.17,0.03,0.17,0.00,0.00,0.00,0.05,
0.00,normal,20

0,icmp,eco_i,SF,8,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,21,0.00,0.00,
0.00,0.00,1.00,0.00,1.00,2,60,1.00,0.00,1.00,0.50,0.00,0.00,0.00,
0.00,ipsweep,17
```

Самое последнее значение в каждой записи в формате CSV – это искусственный результат процедуры улучшения NSL-KDD, который можно не принимать во внимание. Меткой класса является второе с конца значение в каждой записи, а значения с 1 по 41 соответствуют следующим признакам:

```
1 duration
2 protocol_type
3 service
4 flag
5 src_bytes
6 dst_bytes
7 land
8 wrong_fragment
9 urgent
10 hot
22 is_guest_login
23 count
24 srv_count
25 serror_rate
26 srv_serror_rate
27 rerror_rate
28 srv_rerror_rate
29 same_srv_rate
30 diff_srv_rate
31 srv_diff_host_rate
```

---

[1] *Chuong Do and Andrew Ng*. Transfer Learning for Text Classification. Proceedings of the 18th International Conference on Neural Information Processing Systems (2005): 299–306.

[2] *Steffen Bickel*. ECML-PKDD Discovery Challenge 2006 Overview. Proceedings of the ECML-PKDD Discovery Challenge Workshop (2006): 1–9.

[3] *Alexandru Niculescu-Mizil and Rich Caruana*. Inductive Transfer for Bayesian Network Structure Learning. Proceedings of the 11th International Conference on Artiicial Intelligence and Statistics (2007): 339–346.

```
11 num_failed_logins
12 logged_in
13 num_compromised
14 root_shell
15 su_attempted
16 num_root
17 num_file_creations
18 num_shells
19 num_access_files
20 num_outbound_cmds
21 is_host_login
32 dst_host_count
33 dst_host_srv_count
34 dst_host_same_srv_rate
35 dst_host_diff_srv_rate
36 dst_host_same_src_port_rate
37 dst_host_srv_diff_host_rate
38 dst_host_serror_rate
39 dst_host_srv_serror_rate
40 dst_host_rerror_rate
41 dst_host_srv_rerror_rate
```

Наша задача – разработать общий классификатор, который определяет каждый отдельный элемент выборки данных как принадлежащий одному из пяти следующих классов: `benign`, `dos`, `r2l`, `u2r` или `probe`. Тренировочный набор данных содержит элементы выборки, которые снабжены метками конкретной атаки: атаки `ftp_write` и `guess_passwd` соответствуют категории (классу) `r2l`, `smurf` и `updstorm` – категории `dos` и т. д. Отображение (преобразование) меток атак в категории атак определено в файле *training_attack_types.txt*[1]. Выполним некоторое предварительное исследование данных, для того чтобы узнать больше о присвоенных метках. Сначала рассмотрим распределение по категориям[2]:

```
from collections import defaultdict

# Каталог, содержащий все файлы необходимых для работы данных
dataset_root = 'datasets/nsl-kdd'

category = defaultdict(list)
category['benign'].append('normal')

with open(dataset_root + 'training_attack_types.txt', 'r') as f:
    for line in f.readlines():
        attack, cat = line.strip().split(' ')
        category[cat].append(attack)
```

Ниже приведен результат наших исследований содержимого объекта `category`:

```
{
    'benign': ['normal'],
    'probe':  ['nmap', 'ipsweep', 'portsweep', 'satan',
               'mscan', 'saint', 'worm'],
    'r2l':    ['ftp_write', 'guess_passwd', 'snmpguess',
               'imap', 'spy', 'warezclient', 'warezmaster',
               'multihop', 'phf', 'imap', 'named', 'sendmail',
               'xlock', 'xsnoop', 'worm'],
    'u2r':    ['ps', 'buffer_overflow', 'perl', 'rootkit',
               'loadmodule', 'xterm', 'sqlattack', 'httptunnel']
    'dos':    ['apache2', 'back', 'mailbomb', 'processtable',
```

[1] Этого файла нет в наборе данных NSL-KDD, но он включен в исходный набор данных KDD 1999.

[2] Полный исходный код этого примера приведен в форме Python Jupyter notebook в репозитории кода для данной книги в разделе *chapter5/nsl-kdd-classification.ipynb* (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter5/nsl-kdd-classification.ipynb**).

```
                'snmpgetattack', 'teardrop', 'smurf', 'land',
                'neptune', 'pod', 'udpstorm']
    }
```

Обнаружены определения 41 типа атак. Эта структура данных не очень удобна для выполнения преобразования меток атак в категории атак, поэтому необходимо перевести показанный выше словарь в более подходящую форму для дальнейшей обработки:

```
attack_mapping = dict((v,k) for k in category for v in category[k])
```

Теперь структура `attack_mapping` выглядит так:

```
{
    'apache2': 'dos',
    'back': 'dos',
    'guess_passwd': 'r2l',
    'httptunnel': 'u2r',
    'imap': 'r2l',
    ...
}
```

Используемые данные:

```
train_file = os.path.join(dataset_root, 'KDDTrain+.txt')
test_file = os.path.join(dataset_root, 'KDDTest+.txt')
```

Всегда важно принимать во внимание распределение по классам в тренировочном наборе данных и в тестовом наборе. В некоторых случаях точное прогнозирование распределения по классам в реальных данных может оказаться затруднительным, но это полезно для получения общего представления о том, чего следует ожидать. Например, при проектировании классификатора сетевых атак для развертывания в сетевой среде, не содержащей серверов баз данных, можно предположить, что трафик типа `sqlattack` будет минимальным. Такое предположение можно сделать как результат натренированного прогнозирования или на основе предыдущего практического опыта. В нашем конкретном примере есть доступ к тестовым данным, поэтому можно воспользоваться этими данными для оценки их распределения:

```
import pandas as pd

# header_names - это список имен признаков в том же порядке, в котором данным присваивались
# метки (второе с конца значение в записи формата CSV), обозначенные как attack_type,
# а самое последнее значение в CSV-записи обозначено как success_pred

# Чтение тренировочных данных
train_df = pd.read_csv(train_file, names=header_names)
train_df['attack_category'] = train_df['attack_type'] \
        .map(lambda x: attack_mapping[x])
train_df.drop(['success_pred'], axis=1, inplace=True)

# Чтение тестовых данных
test_df = pd.read_csv(train_file, names=header_names)
test_df['attack_category'] = test_df['attack_type'] \
        .map(lambda x: attack_mapping[x])
test_df.drop(['success_pred'], axis=1, inplace=True)
```

Теперь рассмотрим распределения данных по `attack_type` и `attack_category`:

```
import matplotlib.pyplot as plt

train_attack_types = train_df['attack_type'].value_counts()
train_attack_cats = train_df['attack_category'].value_counts()
train_attack_types.plot(kind='barh')
train_attack_cats.plot(kind='barh')
```

На рис. 5.9 и 5.10 показаны распределения тренировочных данных по классам.

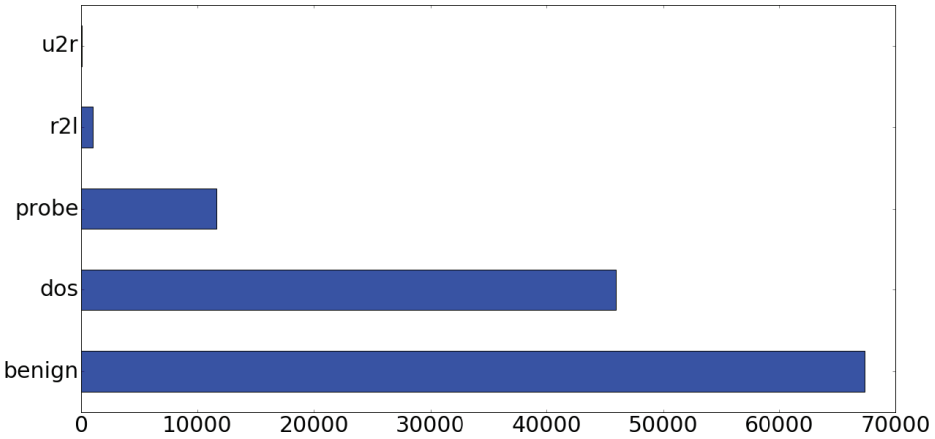


Рис. 5.9 ❖ Распределение тренировочных данных по классам (по пяти категориям)

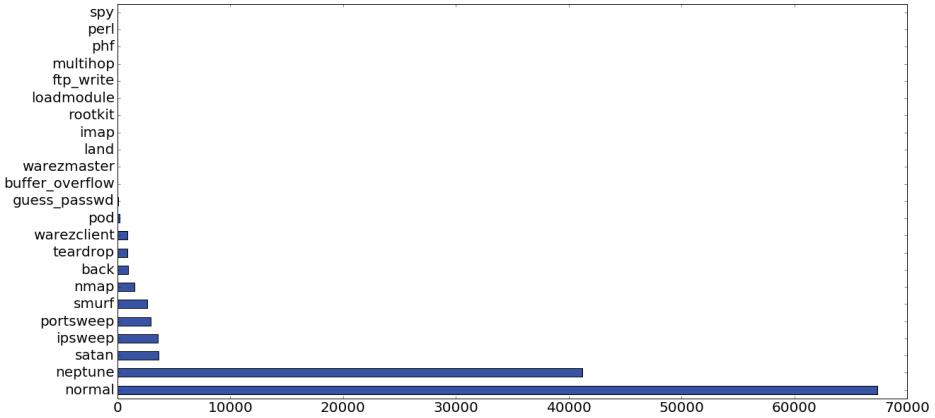


Рис. 5.10 ❖ Распределение тренировочных данных по классам (по 22 категориям)

Тренировочные данные содержат 22 типа трафика атак, а тестовые данные содержат 37 типов трафика атак. Остальные 15 типов трафика атак отсутствуют в тренировочных данных. Набор данных был спроектирован именно так, чтобы проверить, насколько успешно тренируемый классификатор обобщает тренировочные данные. Например, предположим, что классификатор способен правильно классифицировать атаку типа `worm` в категорию `r2l`, даже несмотря на то что

в тренировочных данных не содержатся экземпляры атаки `worm`. Это означает, что такой гипотетический классификатор успешно обучился некоторым обобщенным свойствам категории `r2l`, что позволяет ему правильно классифицировать ранее не встречавшиеся типы атак, относящиеся к этой категории.

В рассматриваемом примере можно явно видеть, что в обоих наборах данных классы абсолютно не сбалансированы. Например, в тренировочном наборе данных размер класса `u2r` на три порядка меньше размера класса `dos`. Если не учитывать этот дисбаланс классов и использовать тренировочные данные в таком виде, то существует вероятность того, что модель «узнает» намного больше о классах `benign` и `dos`, чем о классах `u2r` и `r2l`, в результате чего может возникнуть нежелательное необъективное отклонение в классификаторе. Эта проблема весьма часто возникает в задачах машинного обучения, но для ее устранения существует несколько различных способов. Мы не оставим без внимания столь неприемлемый дисбаланс классов и решим эту проблему несколько позже.

Исходный набор данных 1999 KDD Cup широко использовался для решения задач обеспечения безопасности сети и при исследовании в области выявления вторжений, но подвергался резкой критике со стороны исследователей, сталкивающихся с проблемами в процессе его применения в алгоритмах вычисления оценок. Среди этих проблем выделяется проблема чрезмерной избыточности в тренировочном и тестовом наборах данных, приводящая к искусственно завышаемой точности результатов оценочных алгоритмов, использующих эти данные. В используемом здесь наборе данных NSL-KDD эта проблема устранена. Поводом для критики также стал тот факт, что этот набор данных не является реальным представлением сетевого трафика, поэтому не рекомендуется использовать его в оценочных алгоритмах, предназначенных для развертывания в настоящих рабочих сетевых средах[1]. Эта проблема осталась нерешенной и в наборе данных NSL-KDD. Но поскольку наш пример преследует исключительно учебные цели, не стоит беспокоиться о том, насколько реалистичен этот набор данных, так как мы не применяем оценочные алгоритмы для измерения эффективности работы в реальных сетях. NSL-KDD – это полезный набор данных для учебных целей и для экспериментов в области исследования данных и классификации методами машинного обучения, потому что в нем сохранен баланс между простотой и сложностью. Результаты классификации, получаемые при работе с набором данных NSL-KDD, не так хороши, как результаты, приведенные в других публикациях, описывающих использование исходного набора данных KDD и достижение при этом более 90 % точности классификации.

Подготовка данных

Начнем с разделения тестового и тренировочного наборов данных (DataFrames) на собственно данные и метки:

```
train_Y = train_df['attack_category']
train_x_raw = train_df.drop(['attack_category','attack_type'], axis=1)

test_Y = test_df['attack_category']
test_x_raw = test_df.drop(['attack_category','attack_type'], axis=1)
```

[1] *John McHugh*. Testing Intrusion Detection Systems: A Critique of the 1998 and 1999 DARPA Intrusion Detection System Evaluations as Performed by Lincoln Laboratory. ACM Transactions on Information and System Security 3 (2000): 262–294.

Прежде чем применить какие-либо алгоритмы к этим наборам, необходимо подготовить данные для работы с ними. Во-первых, нужно закодировать категориальные и фиктивные переменные (обозначенные в этом наборе данных как символические переменные). Для удобства сгенерируем список имен категориальных переменных и список имен непрерывных переменных[1]:

```
feature_names = defaultdict(list)

with open(data_dir + 'kddcup.names', 'r') as f:
    for line in f.readlines()[1:]:
        name, nature = line.strip()[:-1].split(': ')
        feature_names[nature].append(name)
```

В результате получаем словарь, содержащий два ключа `continuous` и `symbolic`, причем каждый ключ связан со списком имен признаков:

```
{
    continuous: [ duration, src_bytes, dst_bytes, wrong_fragment, ... ]
    symbolic:   [ protocol_type, service, flag, land, logged_in, ... ]
}
```

Продолжим разделение символических переменных на типы `nominal` (категориальный) и `binary`, поскольку предварительная обработка этих типов будет выполняться по-разному.

Затем воспользуемся функцией `pandas.get_dummies()` (**http://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.get_dummies.html**) для преобразования номинальных переменных в фиктивные переменные. Объединяем наборы `train_x_raw` и `test_x_raw`, пропускаем полученный набор данных через эту функцию, затем снова разделяем на тренировочный и тестовые наборы данных. Такая операция необходима, потому что значения некоторых символических переменных могут присутствовать в одном наборе данных, но не в другом, и отдельная генерация фиктивных переменных для каждого набора может привести к логической несогласованности в столбцах обоих наборов.

Выполнение операций предварительной обработки для объединенного набора тренировочных и тестовых данных обычно не рекомендуется, но в нашем примере это приемлемо. Мы не совершаем никаких действий, которые могут привести к предвзятости применяемого алгоритма, и не смешиваем элементы тренировочного и тестового наборов. В обычных ситуациях мы получили бы полные знания обо всех категориальных переменных, так как мы сами их определили или потому что набор данных предоставляет эту информацию. В нашем случае набор данных не содержит списка возможных значений каждой категориальной переменной, поэтому можно выполнить предварительную обработку следующим образом[2]:

---

[1] Файл *kddcup.names* отсутствует в наборе данных NSL-KDD, но включен в исходный набор данных KDD 1999.

[2] Мы удалили `root_shell` из списка непрерывных признаков, потому что это ошибка в наборе данных. В файле *kddcup.names* `root_shell` неправильно помечен как непрерывный признак, в то время как в документации на этот набор данных четко сказано, что это бинарный признак. Более того, из исследуемых данных становится ясно, что этот признак может принимать только значения 0 и 1. Следовательно, в нашем примере необходимо работать с этим признаком как с бинарным.

```
# Объединение наборов данных DataFrames
combined_df_raw = pd.concat([train_x_raw, test_x_raw])

# Отслеживание непрерывных, бинарных и номинальных признаков
continuous_features = feature_names['continuous']
continuous_features.remove('root_shell')

binary_features = ['land','logged_in','root_shell',
                   'su_attempted','is_host_login',
                   'is_guest_login']

nominal_features = list(
    set(feature_names['symbolic']) - set(binary_features)
)

# Генерация фиктивных переменных
combined_df = pd.get_dummies(combined_df_raw, \
    columns=feature_names['symbolic'], \
    drop_first=True)

# Разделение на тренировочный и тестовый наборы данных
train_x = combined_df[:len(train_x_raw)]
test_x = combined_df[len(train_x_raw):]

# Отслеживание фиктивных переменных
dummy_variables = list(set(train_x)-set(combined_df_raw))
```

В функции `pd.get_dummies()` применяется унитарное, или прямое, кодирование (описанное в главе 2) для категориальных (номинальных) переменных, таких как `flag`, с созданием нескольких бинарных переменных для каждого возможного значения переменной `flag`, которое обнаруживается в исследуемом наборе данных. Например, если элемент выборки имеет значение `flag=S2`, то представление фиктивной переменной (для `flag`) будет следующим:

```
# flag_S0, flag_S1, flag_S2, flag_S3, flag_SF, flag_SH
[    0,       0,       1,       0,       0,       0    ]
```

Для каждого элемента выборки только одна из этих переменных может иметь значение 1, отсюда и название кодирования «унитарное» (one-hot). Как было отмечено в главе 2, для параметра `drop_list` устанавливается значение `True` для предотвращения полной мультиколлинеарности в переменных (ловушка фиктивной переменной) посредством удаления одной переменной из набора сгенерированных фиктивных переменных.

Просматривая распределение признаков в полученном тренировочном наборе, замечаем потенциально опасную особенность:

```
train_x.describe()
```

Таблица 5.1.

	duration	src_bytes	...	hot	num_failed_logins	num_compromised
mean	287.14465	4.556674e+04		0.204409	0.001222	0.279250
std	2604.51531	5.870331e+06		2.149968	0.045239	23.942042
min	0.00000	0.000000e+00		0.000000	0.000000	0.000000
...	...	...		...	...	...
max	42908.00000	1.379964e+09		77.000000	5.000000	7479.000000

Распределения каждого признака изменяются в широких пределах, и это может повлиять на результаты при использовании любых основанных на вычислении расстояний методов для классификации. Например, среднее значение (mean) признака `src_bytes` больше среднего значения признака `num_failed_logins` на семь порядков, а стандартное отклонение (std) больше на восемь порядков. Без проведения стандартизации/нормализации значений признак `src_bytes` будет доминирующим, из-за чего модель может пропустить потенциально важную информацию, содержащуюся в признаке `num_failed_login`.

Стандартизация (standartization) – это процесс, который изменяет масштаб (точнее разброс) последовательности данных, чтобы получить в итоге среднее значение 0 и стандартное отклонение 1 (единичную дисперсию). Это общепринятое, но часто недооцениваемое требование для многих алгоритмов машинного обучения, которое полезно в тех случаях, когда признаки в тренировочном наборе данных имеют широкий диапазон своих характеристик распределения. В библиотеку scikit-learn включен класс `sklearn.preprocessing.StandardScaler` (**https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html**), реализующий требуемую функциональность. Например, попробуем стандартизировать признак `duration`. Как мы только что видели в табл. 5.1, дескриптивные статистические характеристики признака `duration` имеют следующие значения:

```
> train_x['duration'].describe()

count    125973.00000
mean        287.14465
std        2604.51531
min           0.00000
25 %          0.00000
50 %          0.00000
75 %          0.00000
max       42908.00000
```

Применим стандартный метод масштабирования (снижения степени разброса) к этим данным:

```
from sklearn.preprocessing import StandardScaler

# Изменить форму, чтобы сообщить Scaler, что это единственный признак
durations = train_x['duration'].reshape(-1, 1)

standard_scaler = StandardScaler().fit(durations)
standard_scaled_durations = standard_scaler.transform(durations)
pd.Series(standard_scaled_durations.flatten()).describe()

> # Вывод:
count    1.259730e+05
mean     2.549477e-17
std      1.000004e+00
min     -1.102492e-01
25 %     -1.102492e-01
50 %     -1.102492e-01
75 %     -1.102492e-01
max      1.636428e+01
```

Теперь можно видеть, что последовательность промасштабирована и ее среднее значение (mean) чрезвычайно близко к 0 (2,549477 × 10^{-17}), а стандартное отклонение практически равно 1 (1,000004).

Альтернативой стандартизации является нормализация (normalization), при которой изменяется шкала разброса (масштаб) данных – преобразуется в заданный диапазон, обычно [0,1] или [–1,1]. Класс `sklearn.preprocessing.MinMaxScaler` (**https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html**) выполняет нормализацию признака из его исходного диапазона в заданный диапазон `[min, max]`. Если диапазон не задан явно, то по умолчанию принимаются значения `min=0`, `max=1`. Можно воспользоваться классом `MinMaxScaler` вместо `StandardScaler`, если необходима операция масштабирования для сохранения малых стандартных отклонений в исходной последовательности или необходимо сохранить нулевые элементы в разреженных данных. Ниже приведен пример преобразования признака `duration` с помощью класса `MinMaxScaler`:

```
from sklearn.preprocessing import MinMaxScaler

min_max_scaler = MinMaxScaler().fit(durations)
min_max_scaled_durations = min_max_scaler.transform(durations)
pd.Series(min_max_scaled_duration.flatten()).describe()

> # Вывод:
count    125973.000000
mean          0.006692
std           0.060700
min           0.000000
25 %          0.000000
50 %          0.000000
75 %          0.000000
max           1.000000
```

Промахи (выбросы) в исследуемых данных могут значительно исказить результаты стандартизации и нормализации. Если данные содержат промахи, то для работы с ними больше подходит класс `sklearn.preprocessing.RobustScaler` (**https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.RobustScaler.html**). `RobustScaler` использует устойчивые надежные оценки, такие как медиана и интервалы квантилей, поэтому промахи не оказывают значительного воздействия.

При любом варианте выполнения стандартизации или нормализации данных необходимо обязательно применять логически согласованные преобразования и к тренировочному, и к тестовому наборам данных (т. е. использовать одинаковые значения mean, std и др. для преобразования данных). Отдельные операции подгонки одного класса Scaler к тестовому и тренировочному наборам или наличие отдельных классов Scaler для тестовых данных и тренировочных данных является неправильным решением, которое сформирует излишне оптимистичную предвзятость в результатах классификации. При выполнении любого типа предварительной обработки данных следует особое внимание уделить возможной потере информации о тестовом наборе в любой момент времени. Использование тестовых данных для масштабирования тренировочных данных приведет к потере информации о тестовом наборе при выполнении операции тренировки, вследствие чего результаты нельзя считать надежными. Библиотека scikit-learn

предоставляет удобный способ выполнения правильной нормализации для процессов кросс-валидации (перекрестной проверки) – после создания объекта `Scaler` и его подгонки для тренировочных данных можно просто повторно использовать этот же объект для преобразования тестовых данных.

Этап предварительной обработки данных в нашем примере завершается выполнением стандартизации тренировочных и тестовых данных:

```
from sklearn.preprocessing import StandardScaler

# Подгонка класса StandardScaler к тренировочным данным
standard_scaler = StandardScaler().fit(train_x[continuous_features])

# Стандартизация тренировочных данных
train_x[continuous_features] = \
    standard_scaler.transform(train_x[continuous_features])

# Стандартизация тестовых данных с помощью класса Scaler, подогнанного к тренировочным данным
test_x[continuous_features] = \
    standard_scaler.transform(test_x[continuous_features])
```

Классификация

Теперь, когда данные предварительно обработаны и готовы к дальнейшей работе, рассмотрим несколько различных вариантов реальной классификации атак. Продолжим рассмотрение задачи классификации с пятью классами, где каждый элемент выборки принадлежит одному из следующих классов: `benign`, `u2r`, `r2l`, `dos`, `probe`. Существует много различных алгоритмов классификации, подходящих для решения задач такого типа, и множество различных способов для решения задач классификации по нескольким классам. Многие алгоритмы классификации по своей природе поддерживают обработку многоклассовых данных (например, деревья решений, алгоритм ближайших соседей, наивная классификация Байеса, мультиноминальная логистическая регрессия), тогда как другие не поддерживают (например, метод опорных векторов). Даже если выбранный нами алгоритм по своей сущности изначально не поддерживает обработку многих классов, существуют некоторые хитроумные методики выполнения классификации по нескольким классам.

Выбор подходящего классификатора для конкретной задачи может стать одним из наиболее трудных этапов процесса машинного обучения. Для любой задачи существуют десятки классификаторов, и каждый является неплохим выбором, но определить единственный оптимальный вариант весьма сложно. Вообще говоря, специалисты-практики не должны тратить слишком много времени на скрупулезный выбор самого оптимального алгоритма. Разработка решений на основе машинного обучения – итеративный процесс. Затраты времени и усилий на пробные оценочные начальные варианты решения почти всегда дают удивительные практические знания и результаты. Знания и практический опыт работы с различными алгоритмами могут помочь при разработке сразу интуитивно отбросить неэффективные варианты, сократив итерационный цикл, но даже опытные практики редко определяют сразу оптимальный алгоритм, параметры и тренировочную структуру для разнообразных задач машинного обучения. Библиотека scikit-learn предоставляет удобный список с краткими описаниями алгоритмов машинного обучения (**https://s3.amazonaws.com/assets.datacamp.com/blog_assets/**

Scikit_Learn_Cheat_Sheet_Python.pdf) – обзор, помогающий выбрать алгоритм машинного обучения (хотя список неполон). Ниже приведен список общих вопросов, на которые необходимо ответить при выборе наиболее подходящего алгоритма машинного обучения:

- каков размер тренировочного набора данных?
- можно ли спрогнозировать категории выборки или числовые значения?
- имеются ли данные с метками? Сколько данных с метками имеется?
- известно ли количество итоговых категорий?
- каким объемом времени и ресурсов вы располагаете для тренировки модели?
- каким объемом времени и ресурсов вы располагаете для прогнозирования?

В сущности, задачу многоклассовой классификации можно разделить на несколько задач бинарной классификации. Стратегия «один против всех» (one-versus-all), известная также под названием «метод бинарной релевантности» (binary relevance method), определяет по одному классификатору для каждого класса, при этом данные, принадлежащие этому классу, отделяются от всех остальных данных из прочих классов. Немного реже используется стратегия «один против одного» (one-versus-one), согласно которой создается *n_classes* * (*n_classes* – 1) / 2 классификаторов, по одному для каждой неповторяющейся пары классов. В этом случае на этапе прогнозирования каждый элемент выборки проходит через все классификаторы, после чего вычисляются достоверности классификации по каждому из классов. Класс с наибольшей суммарной достоверностью выбирается как результат прогнозирования.

Стратегия «один против всех» масштабируется линейно с ростом количества классов, поэтому в общем обладает лучшей интерпретируемостью модели, поскольку каждый класс представлен только одним классификатором (в отличие от *n_classes* – 1 классификаторов для каждого класса в стратегии «один против одного»). Стратегия «один против одного», напротив, не обеспечивает эффективного масштабирования при росте числа классов из-за присущей ей полиномиальной сложности. Тем не менее стратегия «один против одного» может работать лучше стратегии «один против всех» с классификатором, который масштабируется неэффективно с увеличением количества элементов выборки, потому что классификатор для каждой пары содержит меньшее количество элементов выборки. При стратегии «один против всех» каждый классификатор обязательно должен работать с полным набором данных.

Помимо способности обрабатывать многоклассовые данные, существует множество иных возможных условий, учитываемых при выборе классификатора для конкретной задачи, но подробнее об этом можно узнать из других публикаций[1]. Обладание некоторой долей интуиции в определении соответствия алгоритмов конкретным задачам также может существенно сократить время, затрачиваемое на поиск эффективного решения и оптимизации для получения более надежных результатов. Далее в этом разделе будут рассматриваться примеры применения различных алгоритмов классификации для решения нашей задачи. Несмотря на то что трудно сделать обобщенные выводы о том, как и когда использовать тот

[1] Например, The Elements of Statistical Learning, 2nd ed., by Trevor Hastie, Robert Tibshirani and Jerome Friedman (Springer).

или иной алгоритм, мы все же обсудим некоторые важные условия выбора и применения этих алгоритмов. Применяя параметры, заданные по умолчанию или выбранные интуитивно для каждого алгоритма, можно быстро получить первоначальные результаты классификации. Даже если эти результаты могут оказаться не слишком точными, они обычно позволяют получить общее представление о потенциальных возможностях алгоритма.

Сначала рассматриваются варианты с доступом к тренировочным данным с метками, для которых возможно применение методов классификации (обучения) с учителем. Затем рассматриваются методы классификации с частичным привлечением учителя и классификации без учителя, которые могут применяться вне зависимости от наличия или отстутствия данных с метками.

Обучение с учителем

В нашем распоряжении имеется приблизительно 126 000 элементов тренировочного набора данных с метками, поэтому методы обучения с учителем выглядят неплохим выбором для начала. На практике точность модели не является единственным критерием, который необходимо принимать во внимание. При большом тренировочном наборе эффективность тренировки и время выполнения также являются важными факторами – нежелательно слишком длительное время ожидания результатов первоначального эксперимента с выбранной моделью. Деревья решений или случайные леса – неплохой выбор для начала, поскольку нечувствительны к масштабированию данных (на этапе предварительной обработки) и относительно надежны при работе с неинформативными признаками, следовательно, обычно показывают высокую эффективность тренировки.

Для данных с сотнями измерений использование случайных лесов может оказаться гораздо более эффективным по сравнению с другими алгоритмами с учетом того, насколько хорошо масштабируется эта модель для многомерных данных. Чтобы получить общее представление о работе этого алгоритма, рассмотрим матрицу неточностей (несоответствий), используя sklearn.metrics.confusion_matrix (**https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html**) и коэффициент ошибок sklearn.metrics.zero_one_loss (**https://scikit-learn.org/stable/modules/generated/sklearn.metrics.zero_one_loss.html**). Начнем с простого классификатора – дерева решений sklearn.tree.DecisionTreeClassifier (**https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html**):

```
from sklearn.tree import DecisionTreeClassifier
from sklearn.metrics import confusion_matrix, zero_one_loss

classifier = DecisionTreeClassifier(random_state=0)
classifier.fit(train_x, train_Y)

pred_y = classifier.predict(test_x)

results = confusion_matrix(test_Y, pred_y)
error = zero_one_loss(test_Y, pred_y)

> # Матрица неточностей:
[[9357   59  291    3    1]
 [1467 6071   98    0    0]
 [ 696  214 1511    0    0]
```

```
 [2325    4   16  219   12]
 [ 176    0    2    7   15]]

> # Коэффициент ошибок:
0.238245209368
```

Несколько строк кода без какой-либо предварительной настройки и 76,2 % точности классификации (1 – коэффициент ошибок) в задаче классификации с пятью классами – совсем неплохой результат. Но этот числовой результат почти бесполезен без учета распределений в тестовом наборе (рис. 5.11).

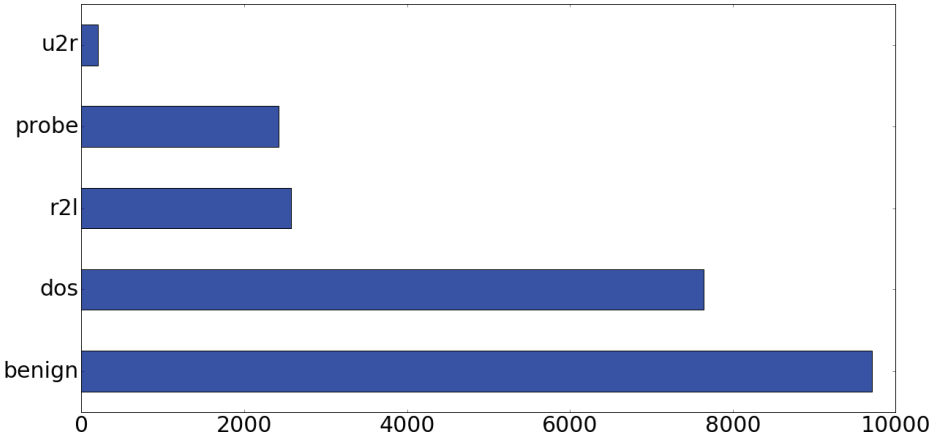


Рис. 5.11 ❖ Распределение по классам в тестовом наборе данных (по пяти категориям)

Матрица неточностей 5×5 может выглядеть несколько пугающе, но хорошее понимание ее содержимого вполне оправдывает дополнительные усилия. В матрице неточностей (несоответствий) значения, расположенные по диагонали (от верхнего левого угла до нижнего правого), – это счетчики правильно классифицированных элементов выборки. Все значения в матрице в сумме дают 22 544 – это размер тестового набора данных. Каждая строка представляет реальный класс, каждый столбец представляет прогнозируемый класс. Например, число в первой строке и пятом столбце представляет количество элементов выборки, которые в действительности относятся к классу 0, но были классифицированы как принадлежащие к классу 4. (Символическим меткам, таким как в нашем примере, методы библиотеки sklearn присваивают числовые значения в непрерывном убывающем порядке: `benign = 0`, `dos = 1`, `probe = 2`, `r2l = 3`, `u2r = 4`.)

Как и в тренировочном наборе, распределение элементов выборки не сбалансировано по категориям:

```
> test_Y.value_counts().apply(lambda x: x/float(len(test_Y)))

benign    0.430758
dos       0.338715
r2l       0.114265
probe     0.107390
u2r       0.008872
```

Здесь можно видеть, что 43 % тестовых данных принадлежат к классу benign, в то время как всего лишь 0,8 % данных относятся к классу u2r. Матрица неточностей показывает, что хотя только 3,7 % элементов класса benign были классифицированы неправильно, 62 % всех элементов тестового набора классифицировались как benign. Похоже, что натренирован классификатор, который больше склонен классифицировать элементы выборки в класс benign. В строках r2l и u2r мы видим проблему в более явно выраженной форме: больше элементов классифицировано как benign, чем все прочие элементы в этих классах. Почему это произошло? Возвращаясь к выполненному ранее анализу, обращаем внимание на то, что только 0,7 % тренировочных данных попали в класс r2l и лишь 0,04 % – в класс u2r. При сравнении с 53,5 % benign, 36,5 % dos и 9,3 % probe кажется, что мы не получили достаточного объема информации для тренировочной модели из классов u2r и r2l для их правильной идентификации. Чтобы узнать, можно ли как-то исправить ситуацию посредством выбора классификатора типа дерева решений, рассмотрим, как работает классификатор по методу k-ближайших соседей sklearn.neighbors.KNeighborsClassifier (**https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html**):

```
from sklearn.neighbors import KNeighborsClassifier

classifier = KNeighborsClassifier(n_neighbors=1, n_jobs=-1)
classifier.fit(train_x, train_Y)
pred_y = classifier.predict(test_x)

results = confusion_matrix(test_Y, pred_y)
error = zero_one_loss(test_Y, pred_y)

> # Матрица неточностей:
[[9455   57  193    2    4]
 [1675 5894   67    0    0]
 [ 668  156 1597    0    0]
 [2346    2   37  151   40]
 [ 177    0    4    6   13]]

> # Коэффициент ошибок:
0.240951029099
```

Ниже приведены результаты работы классификатора по методу линейных опорных векторов sklearn.svm.LinearSVC (**https://scikit-learn.org/stable/modules/generated/sklearn.svm.LinearSVC.html**):

```
from sklearn.svm import LinearSVC

classifier = LinearSVC()
classifier.fit(train_x, train_Y)
pred_y = classifier.predict(test_x)

results = confusion_matrix(test_Y, pred_y)
error = zero_one_loss(test_Y, pred_y)

> # Матрица неточностей:
[[9006  294  405    3    3]
 [1966 5660   10    0    0]
 [ 714  122 1497   88    0]
```

```
 [2464    2    1  109    0]
 [ 175    1    0    8   16]]

> # Коэффициент ошибок:
0.278167139815
```

Коэффициенты ошибки приблизительно одинаковы, а итоговые матрицы неточностей демонстрируют характеристики, очень похожие на характеристики, наблюдавшиеся в результатах классификатора на основе дерева решений. В этот момент должно быть понятно, что продолжение экспериментов с алгоритмами классификации не имеет особого смысла. Здесь можно попытаться как-то смягчить проблему дисбаланса классов в тренировочных данных, надеясь, что итоговые модели не будут слишком искажены смещением в сторону доминирующих классов.

Дисбаланс классов

Работа с несбалансированными классами сама по себе является отдельным профессиональным навыком. В целом можно выделить две категории средств для устранения этой проблемы:

- субдискретизация (undersampling) – выборка из доминирующих классов с целью сокращения в них количества элементов. Стратегии субдискретизации могут быть простыми, как, например, случайный выбор группы элементов, но при этом возможны потери информации в определенных наборах данных. В таких случаях стратегия выборки должна предусматривать в первую очередь удаление элементов, которые очень похожи на другие элементы, остающиеся в наборе данных. Например, при такой стратегии может применяться кластеризация по методу k-средних к большинству элементов класса и удаление точек данных из центроидов с высокой плотностью. Другие, более изощренные методы, например удаление связей Томека (Tomek links)[1], позволяют получить тот же результат. Как и при любых операциях с данными, следует уделять особое внимание побочным эффектам и последствиям субдискретизации и исследовать потенциальные эффекты при слишком большой степени субдискретизации, если наблюдается значительное снижение точности прогнозов для большинства классов;
- выборка с запасом (oversampling) – разумная генерация искусственно создаваемых (синтетических) точек данных для классов с малым количеством элементов представляет собой другой метод устранения неравенства в размерах классов. По своей сущности выборка с запасом полностью противоположна субдискретизации, но это не просто генерация случайных искусственных данных и введение их в класс меньшинства. Требуется внимательность и аккуратность, чтобы не наделить класс характеристиками, которые могут направить процесс обучения по ложному пути. Например, простейшим способом выборки с запасом является добавление случайных элементов в набор данных. Но такой подход, вероятнее всего, загрязнит набор данных и окажет отрицательное воздействие на распределение. Мето-

[1] *Ivan Tomek*. Two Modifications of CNN. IEEE Transactions on Systems, Man and Cybernetics 6 (1976): 769–772.

ды SMOTE[1] и ADASYN[2] являются интеллектуальными алгоритмами, которые пытаются сгенерировать искусственные данные таким способом, который не искажает исходные характеристики класса меньшинства.

Разумеется, при необходимости можно применить сочетание методов субдискретизации и выборки с запасом для взаимной компенсации отрицательных эффектов каждой методики. Широко распространен подход, при котором сначала применяется выборка с запасом для класса меньшинства, затем субдискретизация для снижения уровня несоответствия распределения классов.

Дисбаланс классов является настолько часто возникающей проблемой, что существуют библиотеки ПО, специализирующиеся на решении именно этой задачи. Библиотека imbalanced-learn (**https://imbalanced-learn.readthedocs.io/en/stable/**) предлагает широкий спектр методик повторной выборки данных для решения разнообразных задач. Как и методики классификаторов, различные методики повторной выборки обладают различными характеристиками и могут в той или иной степени подходить для решения конкретных задач. Наилучший способ познакомиться с этими методиками – поэкспериментировать с ними на практике. Отметим, что некоторые из методик, предлагаемых библиотекой imbalanced-learn, не применимы непосредственно к задачам со многими классами. Рассмотрим сначала метод выборки с запасом, применив класс `imblearn.over_sampling.SMOTE` (**https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over_sampling.SMOTE.html?highlight=imblearn.over_sampling.SMOTE**):

```
> print(pd.Series(train_Y).value_counts())

    > # Исходное распределение по классам в тренировочном наборе данных:
        benign    67343
        dos       45927
        probe     11656
        r2l         995
        u2r          52

from imblearn.over_sampling import SMOTE

# Применение метода выборки с запасом SMOTE к тренировочным данным
sm = SMOTE(ratio='auto', random_state=0)
train_x_sm, train_Y_sm = sm.fit_sample(train_x, train_Y)
print(pd.Series(train_Y_sm).value_counts())

    > # распределение по классам в тренировочном наборе данных после первого применения SMOTE:
        benign    67343
        dos       67343
        probe     67343
        u2r       67343
        r2l       67343
```

[1] *N.V. Chawla et al*. SMOTE: Synthetic Minority Over-Sampling Technique. Journal of Artiicial Intelligence Research (2002): 321–357.

[2] *Haibo He et al*. ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced Learning. Proceedings of the IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence) (2008): 1322–1328.

Параметр `ratio='auto'`, передаваемый в конструктор класса `imblearn.over_sampling.SMOTE`, представляет стратегию выборки с запасом, применяемую для выравнивания размеров всех классов меньшинства до размера доминирующего класса.

После нескольких экспериментов мы обнаруживаем, что метод случайной выборки с запасом для используемого в примере тренировочного набора данных дает наилучшие результаты проверки (валидации). Далее нам может помочь класс `imblearn.under_sampling.RandomUnderSampler` (**https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.under_sampling.RandomUnderSampler.html?highlight=imblearn.under_sampling.RandomUnderSampler**):

```
from imblearn.under_sampling import RandomUnderSampler

mean_class_size = int(pd.Series(train_Y).value_counts().sum()/5)

ratio = {'benign': mean_class_size,
         'dos': mean_class_size,
         'probe': mean_class_size,
         'r2l': mean_class_size,
         'u2r': mean_class_size}

rus = RandomUnderSampler(ratio=ratio, random_state=0)
train_x_rus, train_Y_rus = rus.fit_sample(train_x, train_Y)

    > Исходное распределение по классам в тренировочном наборе данных:
          benign    67343
          dos       45927
          probe     11656
          r2l         995
          u2r          52

    > Распределение по классам в тренировочном наборе данных после применения
RandomUnderSampler:
          dos       25194
          r2l       25194
          benign    25194
          probe     25194
          u2r       25194
```

Отметим, что здесь был вычислен средний размер всех 5 классов, который использовался как целевой размер для субдискретизации. Теперь начнем тренировку классификатора на основе дерева решений с помощью предварительно обработанного набора данных и оценим его эффективность:

```
classifier = DecisionTreeClassifier(random_state=17)
classifier.fit(train_x_rus, train_Y_rus)
pred_y = classifier.predict(test_x)

results = confusion_matrix(test_Y, pred_y)
error = zero_one_loss(test_Y, pred_y)

> Матрица несоответствий:
      [[9369   73  258    6    5]
       [1221 5768  647    0    0]
       [ 270  170 1980    1    2]
```

```
       [1829    2  369  369    5]
       [  62    0  108   21    9]]

> Error: 0.223962029808
```

Предварительная обработка тренировочного набора данных помогла достичь уровня точности от 76,2 % до 77,6 %. Это не очень значительное улучшение, но следует принять во внимание, что мы не занимались точной настройкой каждого параметра и не провели достаточного количества экспериментов.

Подбор идеального классификатора для конкретной задачи иногда становится утомительным делом, но может быть также интересным и увлекательным процессом. При проведении ряда экспериментов почти всегда обнаруживаются какие-то неожиданные факты и свойства, и мы узнаем что-то новое о данных, о классификаторе и о самой задаче. Но последовательный перебор всех возможных значений параметров и классификаторов не поможет, если сам набор данных имеет неотъемлемые (присущие от природы) ограничения, такие как недостаток информативных признаков.

Как уже было отмечено выше, генерация тренировочных данных с метками требует дополнительных затрат, поскольку для этого необходимо привлекать экспертов-людей или выполнять определенные физические операции. Поэтому очень часто наборы данных меток не имеют или содержат лишь небольшую часть данных с метками. В таких случаях обучение с учителем может работать, если выбран классификатор, допускающий высокую степень отклонения, такой как наивный байесовский классификатор. Но польза от такого подхода неоднозначна, так как алгоритмы с высоким допуском отклонения по определению подвержены проблеме недоподгонки (underfitting). Тогда возможно перейти к рассмотрению методов обучения с частичным привлечением учителя и методов обучения без учителя.

Обучение с частичным привлечением учителя

Обучение с частичным привлечением учителя – это подкласс алгоритмов машинного обучения с учителем и методов тренировки, специально предназначенных для работы с очень малыми наборами тренировочных данных с метками. Самообучение (self-training)[1] является хорошим примером, которым можно воспользоваться для получения представления о методах обучения с частичным привлечением учителя. Если говорить кратко, эти алгоритмы на первом этапе выполняют вводный процесс тренировки, сначала обучая исходный выбранный механизм оценки на небольшом наборе тренировочных данных, затем начинают работу с данными без меток. Из полученных на начальном этапе результатов извлекаются прогнозы с наибольшей достоверностью и добавляются в тренировочный набор. Этим добавляемым данным присваиваются метки на основе результатов предыдущего раунда прогнозирования. Затем процесс повторяется до тех пор, пока тренировочный набор данных не будет содержать требуемое количество элементов.

---

[1] *Yan Zhou, Murat Kantarcioglu and Bhavani Thuraisingham*. Self-Training with Selection-by-Rejection. Proceedings of the IEEE 12th International Conference on Data Mining (2012): 795–803.

Методика самообучения позволяет сформировать тренировочный набор данных разумного размера, который в большинстве случаев вполне соответствует требованиям по качеству. Некоторые другие методики обучения с частичным привлечением учителя основаны на обобщенных способах формирования данных с искусственно созданными метками из существующих выборок (это почти та же концепция, на которой основана методика выборки с запасом, описанная в предыдущем разделе) или на методах кластеризации по плотности и на графовых алгоритмах для обучения на разнообразных наборах данных. Эти методики позволяют создавать (путем логических выводов) метки для изначально не помеченных точек данных с достаточно обоснованной степенью достоверности.

Библиотека scikit-learn предоставляет классы `sklearn.semi_supervised.LabelPropagation` (**https://scikit-learn.org/stable/modules/generated/sklearn.semi_supervised.LabelPropagation.html**) и `sklearn.semi_supervised.LabelSpreading` (**https://scikit-learn.org/stable/modules/generated/sklearn.semi_supervised.LabelSpreading.html**), которые могут помочь в улучшении решений на основе методов обучения с частичным привлечением учителя[1].

Обучение без учителя

Что делать, если данных с метками вообще нет? В некоторых ситуациях, возникающих при обеспечении безопасности, достаточно часто очень трудно сгенерировать тренировочные данные с метками (например, анализ бинарных файлов требует многочасовой работы по поиску характерных следов и исследованию каждого элемента выборки, а расследование конкретных инцидентов требует огромного объема ресурсов и бюрократической организации делопроизводства на нескольких уровнях). В таких случаях единственным подходящим вариантом является обучение без учителя. Этот мощный статистический метод выводит скрытую структуру из тренировочных данных без меток. В области классификации явно преобладающим классом методов обучения без учителя является кластеризация (clustering).

Как уже отмечалось в предыдущих главах, к кластеризации относятся методики объединения в группы схожих элементов данных по некоторому определению схожести. Каждая группа похожих друг на друга точек называется кластером (cluster), а каждый кластер представляет модель, обучаемую по некоторой категории. Существуют десятки моделей кластеризации, из которых любая может применяться в различных ситуациях. Некоторые методы требуют знания количества классов или центроидов заранее (например, метод k-средних, смешанные модели Гаусса (модели Гаусса со смешанными распределениями)), для других это не обязательно (например, для кластеризации методом сдвига среднего значения). Это различие является наиболее важным фактором при выборе метода кластеризации для конкретной задачи.

---

[1] В нашем примере не используются методы обучения с частичным привлечением учителя, потому что для их практического применения обычно требуется долговременный процесс анализа сходства между кластерами, весьма специфический для каждого используемого набора данных. Для более подробного изучения методов обучения с частичным привлечением учителя рекомендуется книга Semi-Supervised Learning by Olivier Chapelle, Bernhard Schölkopf and Alexander Zien (MIT Press).

Выполнение небольшого примера кластеризации для набора данных NSL-KDD позволит нам сразу же увидеть, чем кластеризация отличается от методик обучения с учителем, рассмотренных в предыдущем разделе. В нашем примере известно, что данные содержат пять категорий элементов, поэтому выбирается алгоритм кластеризации методом k-средних для задачи с k (число кластеров), равным 5. (DBSCAN[1] является другим часто выбираемым вариантом кластеризации, но больше подходит для данных, содержащих кластеры приблизительно одинаковой плотности, поэтому для нашей задачи он не пригоден.) Отметим, что метод k-средних работает только с непрерывными признаками (подробнее об этом *см*. главу 2), поэтому в наборе данных для примера используются лишь непрерывные признаки. Воспользуемся классом `sklearn.cluster.KMeans` (**https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html**):

```
from sklearn.cluster import KMeans

kmeans = KMeans(n_clusters=5).fit(train_x)
pred_y = kmeans.predict(test_x)

# Результаты кластеризации
print(pd.Series(pred_y).value_counts())

>
1    15262
2     5211
0     2069
3        2
```

Результат заметно отличается от результатов, полученных по методикам обучения с учителем. Результатом кластеризации по методу k-средних является определенное количество кластеров, при этом каждый кластер помечен произвольно выбранным индексом. В рассматриваемом примере отметим, что тестовый набор содержит только четыре кластера, хотя изначально был определен параметр `n_clusters=5`. По результату видно, что в самый последний кластер не включено ни одной точки из тестового набора данных, но этого можно было ожидать, учитывая обсуждение дисбаланса классов в одном из предыдущих разделов. Гораздо интереснее оценить качество полученных результатов алгоритмов кластеризации. Ни одному из кластеров не была присвоена метка на основании какой-либо переданной в алгоритм предварительной информации (в этом нет необходимости). Таким образом, оценка результатов кластеризации – это не простое сравнение ожидаемых и прогнозируемых меток.

Для оценки выбранной модели вычисляется, сколько элементов `benign` сгруппировано в одном кластере и сколько элементов других классов помещено в этот же кластер. Здесь вычисляются две метрики: коэффициент полноты (completeness score) и коэффициент гомогенности (homogenity score). Кластер является полным (его коэффициент полноты равен 1), если все точки данных, принадлежащие к одному классу (т. е. имеющие одинаковые метки класса), объединены в этом

[1] В главе 2 метод DBSCAN описан более подробно. См. также *Martin Ester et al*. A Density-Based Algorithm for Discovering Clusters in Large Spatial Databases with Noise. Proceedings of the 2nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (1996): 226–231.

кластере. С другой стороны, кластер является гомогенным (его коэффициент гомогенности равен 1), если все точки данных, сгруппированные в этом кластере, принадлежат к одному классу. V-мера (V-measure), определяемая как гармоническое среднее значение коэффициентов гомогенности и полноты, представляет собой удобную единую метрику для оценки результата:

```
v_measure_score = 2 * (homogeneity * completeness) / (homogeneity + completeness)
```

Определим числовые оценки для результатов кластеризации по методу k-средних, примененному к набору данных NSL-KDD, как показано ниже:

```
from sklearn.metrics import completeness_score,\
    homogeneity_score, v_measure_score

print('Completeness: {}'.format(completeness_score(test_Y, pred_y)))
print('Homogeneity: {}'.format(homogeneity_score(test_Y, pred_y)))
print('V-measure: {}'.format(v_measure_score(test_Y, pred_y)))
```

```
> Completeness: 0.403815151707
> Homogeneity: 0.263893938589
> V-measure: 0.319194009471
```

Значение V-меры 0,32 – это очень плохой результат. Может показаться, что данные в их текущем состоянии совершенно не подходят для классификации без учителя. Действительно, дисбаланс классов создает большую проблему при кластеризации, где алгоритмы рассчитаны на общие признаки точек данных для формирования (в идеальном случае) плотных кластеров, состоящих из похожих друг на друга точек. Недостаток информации о классах, представленных минимальным количеством элементов, может привести к совершенно некорректному формированию кластеров меньшинства, когда из правильно сформированных кластеров «похищаются» точки данных.

Эксперименты с методиками повторной выборки (ресэмплинга) данных могут помочь в устранении подобных проблем, но здесь важно отметить, что некоторые типы задач по своей природе не подходят для обучения без учителя. Если данные не содержат явно выделяемые кластеры элементов, то кластеризация становится сложнейшей задачей. Как можно определить соответствие и несоответствие? Визуализация всегда полезна для полного понимания исследуемых данных, но особенно полезна при кластеризации. Необходимо изобразить элементы данных в некотором пространстве и визуально оценить возможность формирования кластеров. Перед попыткой отображения данных в графической форме проверим, сколько измерений/признаков имеется в нашем распоряжении:

```
> print('Total number of features: {}'.format(len(train_x.columns)))
```

```
Total number of features: 119
```

Вряд ли возможно графически изобразить все 119 измерений, поэтому выполним операцию снижения размерности (dimensionality reduction) для преобразования данных в набор с двумя измерениями, более удобный для изображения в двумерной декартовой системе координат. Чаще всего для снижения размерности применяется метод главных компонентов (principal component analysis – PCA)[1].

[1] *Svante Wold, Kim Esbensen and Paul Geladi*. Principal Component Analysis. Chemometrics and Intelligent Laboratory Systems 2 (1987): 37–52.

Этот процесс без подробных математических формулировок и выкладок точно объяснить очень трудно, но можно отметить, что метод PCA эффективно находит в многомерном пространстве набор осей (главных компонентов), который упорядочен по убыванию значений дисперсии в описываемых данных (от максимума к минимуму). Набор данных отображается на первые несколько осей, обеспечивающих наибольший объем информации в этом наборе.

Выполнение операции снижения размерности методом PCA в действительности так же просто, как выбор n самых главных компонентов и представление исследуемых данных в этих n измерениях. В большинстве случаев практического применения метода PCA с целью снижения размерности для нисходящей классификации или обработки необходимо выбрать достаточное число главных компонентов для охвата большей части дисперсии в исследуемых данных. Но при необходимости визуального отображения чаще всего вполне достаточно выбрать два самых главных компонента для успешного представления и понимания сущности данных. Для нашего тренировочного набора данных воспользуемся классом `sklearn.decomposition.PCA` (**https://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html**), помечающим точки данных эмпирическими метками (рис. 5.12):

```
from sklearn.decomposition import PCA

pca = PCA(n_components=2)
train_x_pca = pca.fit_transform(train_x)

plt.figure()
colors = ['navy', 'turquoise', 'darkorange', 'red', 'purple']

for color, cat in zip(colors, category.keys()):
    plt.scatter(train_x_pca[train_Y==cat, 0],
   train_x_pca[train_Y==cat, 1],
                color=color, alpha=.8, lw=2, label=cat)
plt.legend(loc='best', shadow=False, scatterpoints=1)
plt.show()
```

Первый главный компонент представлен по горизонтальной оси, второй – по вертикальной оси. Сразу становится ясно, что этот набор данных не очень подходит для кластеризации. Выборки по классам `probe`, `dos` и `r2l` распределены непредсказуемым беспорядочным образом и не образуют четко выраженных кластеров. Элементов класса `u2r` очень мало, поэтому этот класс трудно выделить на графике, он не образует заметных кластеров. Хорошо видно, что только элементы класса `benign` формируют четко определенный кластер. Из исследовательского интереса построим график распределения тех же точек с метками, присвоенными в результате подгонки данных к алгоритму кластеризации по методу k-средних, который использовался ранее:

```
from sklearn.cluster import KMeans

# Подгонка тренировочных данных к модели оценки кластеризации методом k-средних
kmeans = KMeans(n_clusters=5, random_state=17).fit(train_x)

# Извлечение меток, присвоенных каждому элементу тренировочного набора
kmeans_y = kmeans.labels_
```

```
# Построение двумерного графика с помощью train_x_pca_cont
plt.figure()
colors = ['navy', 'turquoise', 'darkorange', 'red', 'purple']

for color, cat in zip(colors, range(5)):
    plt.scatter(train_x_pca_cont[kmeans_y==cat, 0],
                train_x_pca_cont[kmeans_y==cat, 1],
                color=color, alpha=.8, lw=2, label=cat)
plt.legend(loc='best', shadow=False, scatterpoints=1)

plt.show()
```

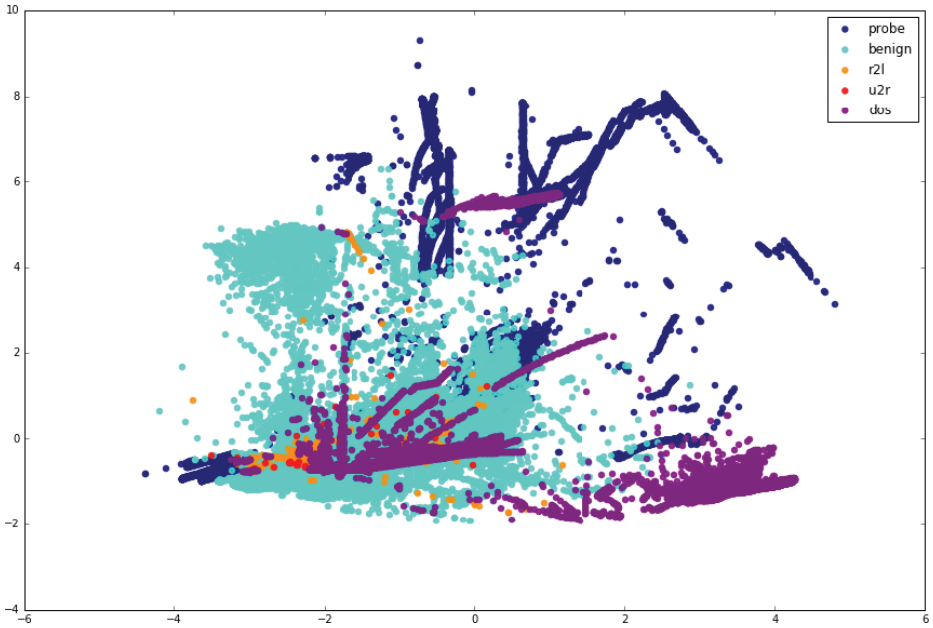


Рис. 5.12 ❖ График распределения тренировочных данных с эмпирическими метками (преобразование методом PCA)

На рис. 5.13 показан результат.

Теперь напомним, что алгоритм k-средних относится к группе методов обучения без учителя, поэтому неизвестно, какие именно присваиваемые метки соответствуют каждому формируемому кластеру. На рис. 5.13 сразу можно видеть существенные различия между кластерами, сформированными по методу k-средних, и кластерами с эмпирическими метками. Алгоритм хорошо работает при группировании явно выраженных областей данных, но ошибается при формировании кластеров по метке `dos` и неправильно классифицирует большую группу данных «легального» трафика как трафик атаки (верхняя левая область на рис. 5.13). Точная настройка и экспериментирование со значением k могут помочь в устранении этой проблемы, но даже по результатам столь краткого визуального анализа можно видеть, что здесь, вероятнее всего, существуют более глубокие проблемы с применением одних только методов кластеризации для классификации сетевых атак по имеющимся данным.

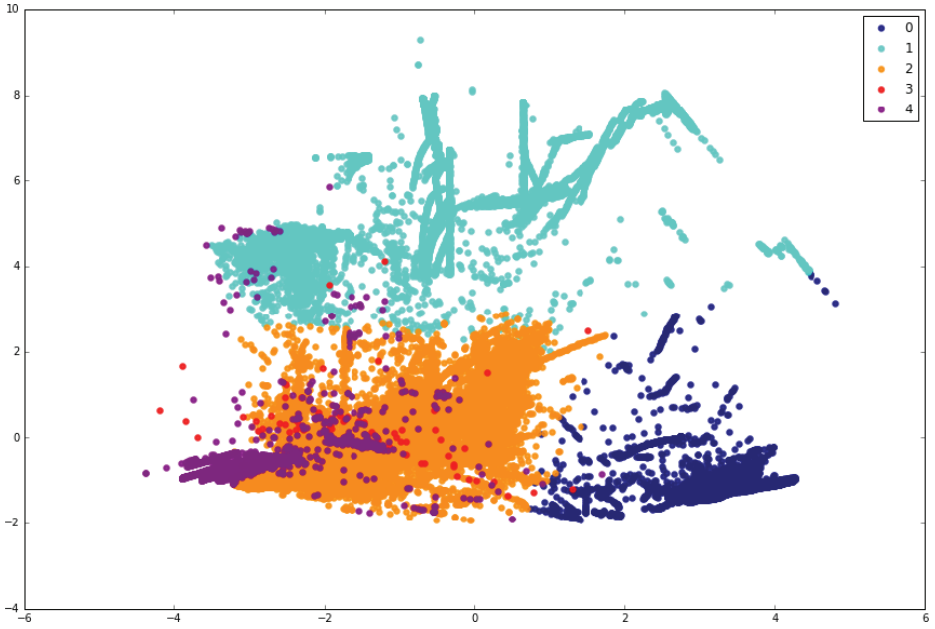


Рис. 5.13 ❖ График распределения тренировочных данных с прогнозируемыми по методу k-средних классами (k = 5)

Даже если использовать только непрерывные признаки, придется обрабатывать набор данных с 34 измерениями. Из главы 2 нам известно, что на наборах данных с высокими размерностями эффективность метода k-средних снижается. Проблему можно устранить, предоставив алгоритму данные со сниженной размерностью (например, методом PCA). Но гораздо труднее решить проблему присущего исследуемому набору данных распределения характеристик. В главе 2 также отмечалось, что метод k-средних не вполне подходит для распределений, отличающихся от сферических. По рис. 5.12 трудно утверждать, что какой-либо класс имеет сферическое распределение. Поэтому нас не должно удивлять, что метод k-средних в нашем примере показал неудовлетворительный результат.

Расширенное ансамблирование

Метод k-средних относительно хорошо работает с предварительно присвоенными метками в непрерывных кластерах данных. Попытка применения метода k-средних с значением k = 8 дает результат, показанный на рис. 5.14.

При сравнении этого графика с распределением эмпирических меток на рис. 5.12 можно заметить, что некоторые кластеры данных в основном принадлежат одному классу. Другие кластеры, например кластер 2, сформированы из смешанных данных, соответствующих типам трафика `benign` и `probe`. Если бы существовал способ выполнения дальнейшей классификации внутри кластеров, возможно, нам удалось бы добиться более успешных результатов. Такое предположение вполне естественно – действительно, существует широко известная группа методик, называемая ансамблированием (ensembling).

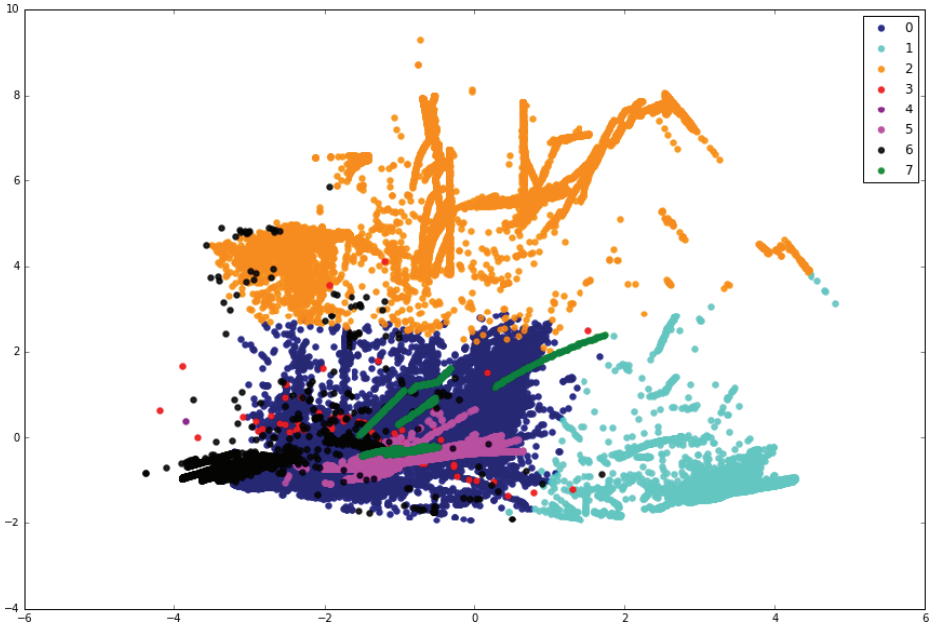


Рис. 5.14 ❖ График распределения тренировочных данных с прогнозируемыми по методу k-средних классами (k = 8)

Методики ансамблирования или модели ансамблей (ensemble models) подразумевают объединение двух и более моделей машинного обучения (которые могут использовать один и тот же внутренний алгоритм или абсолютно различные алгоритмы) для формирования единой системы[1]. Цель такого объединения – получение совокупности результатов работы нескольких слабых моделей для формирования единого мощного механизма обучения.

В рассматриваемом примере мы пытаемся выполнить дальнейшую классификацию отдельно в некоторых кластерах, которые содержат смешанные метки, чтобы попытаться разделить эти кластеры более точно, чем это можно было бы сделать методом k-средних. Для упрощения задачи попробуем классифицировать весь сетевой трафик по двум наиболее обобщенным классам: трафик `attack` и трафик `benign`. Мы снизили размерность задачи, используя метод ранжирования признаков, называемый также отношением атрибутов или характеристик (attribute ratio)[2]. Здесь мы не рассматриваем подробности этого метода[3], но применяем ранжирование для выбора признаков с отношением атрибутов (характеристик),

---

[1] Здесь мы не рассматриваем подробно методики ансамблирования. Для более глубокого изучения ансамблирования можно рекомендовать книгу Ensemble Machine Learning: Methods and Applications by Cha Zhang and Yunqian Ma (Springer Publishing).

[2] *Hee-su Chae and Sang Hyun Choi*. Feature Selection for Efficient Intrusion Detection Using Attribute Ratio. International Journal of Computers and Communications 8 (2014): 134–139.

[3] Полное описание и реализация методики вычисления отношений атрибутов приведены в разделе «Using “Attribute Ratio” (AR) feature selection» Python Jupyter notebook в репозитории кода **https://github.com/oreilly-mlsec/book-resources/blob/master/chapter5/nsl-kdd-classification.ipynb** для данной книги.

большим 0,01. Это позволяет сократить количество используемых признаков до 31. Сначала определим метки для нашей новой задачи:

```
train_Y_bin = train_Y.apply(lambda x: 'benign' if x == 'benign' else 'attack')
```

Затем выбираем подмножество наиболее важных признаков, применяем к нему метод k-средних с k = 8 и составляем таблицу сопряженности или таблицу контингентности (cross-tabulation) восьми кластеров и эмпирических бинарных меток (табл. 5.2):

```
kmeans = KMeans(n_clusters=8, random_state=17).fit(
    train_df_ar_trimmed[continuous_features_trimmed])
kmeans_train_y = kmeans.labels_

pd.crosstab(kmeans_train_y, train_Y_bin)
```

Таблица 5.2. Таблица сопряженности кластеров, спрогнозированных по методу k-средних, и тренировочных данных, размеченных эмпирически

Категории атаки	attack	benign
0	6457	63 569
1	11 443	2784
2	34 700	126
3	0	1
4	4335	628
5	757	167
6	884	0
7	54	68

Рассматривая табл. 5.2, необходимо определить, как предполагается обрабатывать каждый кластер. Отметим, что в кластерах 2 и 6 наблюдается явное преобладание трафика атак, в кластерах 3 и 7 очевиден чрезвычайно низкий уровень «шума», а в остальных кластерах содержатся смешанные данные. Разработка оптимальной стратегии объединения на основе тренировочных данных является важнейшим этапом ансамблирования, поэтому вполне оправданы значительные затраты времени на эксперименты с целью определения самой эффективной методики. В рассматриваемом примере применяется трехэтапная стратегия для обработки каждого кластера.

1. Для кластеров с суммарным размером менее 200 элементов учитываются промахи, и им присваивается метка `attack`.
2. Для кластеров, в которых более 99 % элементов принадлежат одному классу (либо `attack`, либо `benign`), всем элементам присваивается доминирующая метка.
3. Для всех прочих кластеров тренируется классификатор на основе дерева решения (отдельное для каждого кластера).

В соответствии с этой стратегией кластеры 3 и 7 будут считаться «зашумленными», и им присваивается метка `attack`. В кластерах 2 и 6 более 99 % трафика классифицировано как `attack`, поэтому метка распространяется на весь кластер. Для каждого кластера 0, 1, 4 и 5 будет выполняться тренировка отдельного классификатора на основе дерева решений.

Методика оценивается по первому прогону тестового набора через модель k-средних для получения тестовых кластеров. Затем все тестовые выборки, распределенные в кластеры 2, 3, 6 и 7, сразу же будут помечены как `attack`. Тестовые выборки, отнесенные к кластеру 0, будут обработаны классификаторами на основе дерева решений, тренированными специально для кластера 0 с использованием тренировочного набора, и т. д.

В качестве примера продемонстрируем процесс тренировки классификатора на основе дерева решений для кластера 4 и прогнозирование по тестовой выборке кластера 4:

```
train_y4 = train_df[train_df.kmeans_y == 4]
test_y4 = test_df[test_df.kmeans_y == 4]

dfc4 = DecisionTreeClassifier(random_state=17)
.fit(train_y4.drop(['kmeans_y'], axis=1),train_y4['labels2'])

dtc4_pred_y = dtc4.predict(test_y4.drop(['kmeans_y'], axis=1))
```

В соответствии с выбранной стратегией объединения в табл. 5.3 показан общий результат в виде матрицы несоответствий для всех восьми кластеров.

Таблица 5.3. Матрица несоответствий для объединенной модели ансамбля классификаций

	Прогнозируемые метки benign	**Прогнозируемые метки attack**
Правильные метки benign	9418	293
Правильные метки attack	4020	8813

При анализе отчета о классификации получаем результаты, показанные в табл. 5.4.

Таблица 5.4. Статистические характеристики модели ансамбля классификаций

	Точность	**Полнота**	**Метрика F_1**	**Носитель меры**
benign	0.70	0.97	0.81	9711
attack	0.97	0.69	0.80	12 833

Здесь:

Точность = Истинно положительные результаты / (Истинно положительные результаты + Истинно отрицательные результаты);

Полнота = Истинно положительные результаты / (Истинно положительные результаты + Ложноотрицательные результаты).

Другими словами, точность – это относительное количество правильно спрогнозированных элементов, являющихся релевантными, а полнота – это относительное количество релевантных элементов, которые правильно спрогнозированы.

Можно видеть, что точность прогнозирования атак равна 97 %. Это означает, что если классификатор определяет (прогнозирует) элемент как атаку, то с вероятностью 97 % это действительно атака. Полнота для трафика `benign` также равна 97 %, т. е. 97 % всего действительно легального трафика правильно определено (спрогнозировано) как трафик типа `benign`.

Проблемы наблюдаются в сравнительно низкой точности прогнозирования легального трафика и малой полноте трафика атак. Фактически 30 % трафика, определенного классификатором как легальный, в действительности является атакующим.

Как объясняются столь неудачные результаты прогнозирования легального трафика? Рассмотрим таблицу сопряженности прогнозов по методу k-средних для тестового набора данных и сравним ее с результатами по тренировочным данным:

```
# kmeans_test_y - кластеры, спрогнозированные по методу k-средних
pd.crosstab(kmeans_test_y, test_Y_bin)
```

В табл. 5.5 показаны результаты.

Таблица 5.5. Таблица сопряженности прогнозов по методу k-средних для кластеров и эмпирических тестовых данных

Категории атак	Трафик атаки	Легальный трафик
0	9515	4795
1	87	5131
2	6	1997
3	0	0
4	51	427
5	10	1
6	37	8
7	5	474

Если сравнить табл. 5.1 с табл. 5.5, то можно заметить, что наибольшие расхождения между распределениями `attack/benign` имеются в кластере 0[1]. В тренировочных данных кластер 0 содержит только 9 % атакующего трафика, в то время как в тестовых данных доля атакующего трафика в кластере 0 составляет 66 %. Есть основания предположить, что информации, извлеченной моделью k-средних для кластера 0, недостаточно для распространения обобщения на тестовые данные. Подтвердим это предположение исследованием матрицы несоответствий для прогнозов классификатора на основе дерева решений для кластера 0, показанной в табл. 5.6.

Таблица 5.6. Матрица несоответствий только для кластера 0 (классификатор на основе дерева решений)

	Прогнозируемый легальный трафик	Прогнозируемый трафик атак
Действительный легальный трафик	9352	163
Действительный трафик атак	3062	1733

[1] Нам известно, что значительные различия в распределении меток также наблюдаются в кластерах 5, 6 и 7, но количество различающихся элементов в этих кластерах относительно мало по сравнению с общим размером набора данных. Следовательно, такие небольшие расхождения можно объяснить шумом в наборе данных.

При сравнении табл. 5.3 и табл. 5.6 отметим, что 76 % (3062 из 4020) всех ложноотрицательных прогнозов (спрогнозированный легальный трафик + действительный трафик атак), сделанных ансамблем, являются результатом работы классификатора на основе дерева решений кластера 0. Если бы можно было улучшить классификатор кластера 0, то у нас появилась бы возможность существенно повысить общую эффективность ансамбля классификаторов.

Теперь может возникнуть соблазн сразу же применить результаты этого конкретного теста для улучшения используемой модели. Возможно, удастся улучшить результаты, применив другой алгоритм только для кластера 0? В большинстве случаев такой способ специализированной настройки может стать главной ошибкой при практическом использовании машинного обучения. Никогда не следует вносить усовершенствования в модель, основываясь только на результатах работы с тестовым набором данных.

Тем не менее рассмотрим следующую ситуацию: предположим, что выбранная модель реально эксплуатируется в течение месяца, после чего вы замечаете, что классификатор на основе дерева решений для кластера 0 постоянно демонстрирует неэффективность. Внесение изменений в модель на основе такой информации вполне обосновано, более того, настоятельно рекомендуется.

В чем различие? Отметим, что в последнем случае результаты обработки тренировочного набора были распространены на данные, реально собранные (и, возможно, помеченные соответствующим образом) в течение месяца. Теперь вы можете использовать такой заново переопределенный тренировочный набор для разработки следующей версии классификатора. Но при этом обязательно следует помнить и о необходимости генерации тестового набора, соответствующего новому тренировочному набору данных.

Резюме

При практической работе с примером решения конкретной задачи в этой главе должно прийти полное понимание того, что машинное обучение, особенно в отношении корреляции и классификации данных, является не просто процедурой выбора правильного классификатора и не ограничивается знанием алгоритмов. Затраты времени на исследование и понимание природы данных необходимы и чрезвычайно важны, потому что помогают сберечь драгоценное время для достижения требуемой точности классификации. В сфере обеспечения безопасности предлагаются действительно интересные и трудные задачи, для которых необходимо применять методы машинного обучения. В большинстве случаев приходится иметь дело с дисбалансом классов и недостаточным объемом тренировочных данных. Иногда продолжительная точная настройка алгоритмов и процесс обучения могут не дать желаемого результата. В этом случае сбор бóльших объемов данных, генерация более описательных признаков, изменение определений классов/категорий, корректировка целей обучения или все перечисленные меры в совокупности, возможно, станут именно теми необходимыми средствами, которые позволят получить наилучший результат.

Глава 6

Защита потребительской веб-среды

До настоящего момента почти все наше внимание было сосредоточено на предотвращении вредоносных действий злоумышленников и их вторжений в операционную среду компьютеров или в сетевую среду, на их обнаружении после взлома и проникновения через уязвимые места, а также на снижение влияния уязвимостей на систему защиты. Но атакующему далеко не всегда необходимо внедриться в сетевую среду или в операционную среду компьютера, чтобы извлечь какие-то экономические (финансовые) выгоды. В этой главе рассматриваются атаки с использованием внешних общедоступных интерфейсов потребительских веб-сайтов или функциональность пользовательских приложений для достижения корыстных целей.

При использовании термина «потребительская веб-среда» (consumer web) подразумевается любой сервис, доступный через обычный открытый для всех интернет, который предоставляет продукцию или услуги индивидуальным потребителям. Такой сервис может быть бесплатным или оплачиваемым. Здесь необходимо отличать потребительскую веб-среду от корпоративных сервисов, предоставляемых только конкретным организациям, а также от внутренних сетевых сред компаний и организаций.

В потребительской веб-среде существует множество различных поверхностей атак, в том числе доступ к учетным записям, платежные интерфейсы и средства генерации контента. Социальные сети образуют дополнительное измерение (пространство) уязвимости, поскольку атакующие могут воспользоваться всеми преимуществами социального графа (social graph) для достижения своих целей.

Но потребительская веб-среда обладает также и некоторыми неотъемлемыми свойствами, предоставляющими определенные преимущества защищающейся стороне. Главным свойством является крупномасштабность: любой сайт, являющийся целью атаки, также является целью гораздо более мощного по объему легального трафика. Это означает, что при создании защиты имеется большая база данных легальных шаблонов и образцов, которые можно использовать для тренировки выбранных алгоритмов. Здесь вполне может подойти методика выявления аномалий, рассмотренная в главе 3, особенно если в вашем распоряжении нет большого объема данных с метками. С другой стороны, если рассматривается вариант встраивания средств защиты непосредственно в программный продукт, возможно, потому что ваш веб-сайт или приложение уже подвергались атакам,

то в этом случае, вероятнее всего, у вас уже накоплена достаточно крупная база данных с метками для создания классификатора, обучаемого с учителем.

В последующих разделах будут рассматриваться разнообразные векторы атак и методы, которыми машинное обучение может помочь отличить легальную деятельность от вредоносных действий в каждом конкретном случае. Поскольку в сообществе ПО с открытым исходным кодом мошенничество с данными встречается редко, главное внимание будет сосредоточено на основополагающих принципах генерации признаков и выбора алгоритма для каждой задачи. На протяжении всей главы будет подробно рассматриваться конкретный пример, демонстрирующий различные методики кластеризации.

Несмотря на то что эта глава называется «Защита потребительской веб-среды», весь излагаемый в ней материал в равной степени применим и к веб-сайтам, доступным через браузеры и приложения, получающим данные от API, обеспечивающих взаимодействие с интернетом. В следующих разделах мы рассмотрим некоторые различия в поверхностях атак и признаках, доступных защищающейся стороне.

Монетизация в потребительской веб-среде

Многие ориентированные на потребителя веб-сайты предоставляют злоумышленникам возможности монетизации просто при прямом доступе к любой учетной записи. Самый очевидный пример – финансовые организации, но онлайновые торговые площадки, сервисы совместного использования автомобилей, рекламные сети и даже отели и авиакомпании предлагают бонусные программы, которые также выполняют операции прямой передачи денежных средств абсолютно случайным клиентам. Злоумышленникам достаточно лишь взломать несколько учетных записей с крупными финансовыми операциями, для того чтобы полностью окупить все затраты. Таким образом, защита учетных записей на подобных сервисах становится главным и непременным условием дальнейшего существования сайта.

Мошенничество является еще одной крупнейшей проблемой для многих веб-сайтов, ориентированных на потребителя. Большинство онлайновых сервисов принимает кредитные карты для оплаты, а злоумышленники могут похитить кредитные карты для оплаты продукции, предлагаемой на таких сайтах. Эта опасность особенно велика на сайтах, напрямую предлагающих потребительские товары, поскольку полученные нечестным путем товары в дальнейшем могут быть перепроданы на черных рынках по более низким ценам. Угроза мошенничества существует даже для сайтов, предлагающих цифровую продукцию, поскольку и цифровая продукция может быть перепродана. Кроме того, учетные записи с поощрением особо ценных клиентов также могут использоваться для распространения мошеннической деятельности других типов на таких сервисах.

Мошенничество является сферой деятельности, более широкой, нежели простое присваивание платежей. Кликфрод (click fraud) состоит из ботов и других инструментальных средств автоматизации, выполняющих клики (щелчки, переходы) по ссылкам с единственной целью – увеличение соответствующего счетчика кликов. Чаще всего кликфрод встречается в рекламной сфере для увеличения собственного дохода или для истощения рекламных бюджетов конкурентов. Но

кликфрод может появляться в любых местах, где от счетчиков зависит рейтинг или прибыль, как, например, количество просмотров некоторого видеофрагмента или количество «лайков» у некоторого сообщения (поста) пользователя. Мы также рассматриваем рейтинговое мошенничество (review fraud), при котором пользователи создают предвзятые рецензии на продукцию или услуги, чтобы искусственно завысить или понизить рейтинг продукции либо услуги. Например, владелец ресторана может создавать (или оплачивать сторонним лицам создание) ложные «пятизведочные» рецензии (отзывы) о своем ресторане на справочном сайте для привлечения новых клиентов.

Даже если злоумышленник не может напрямую монетизировать свои действия, существует большое количество сайтов, которые предоставляют средства косвенной монетизации. Спам – самое распространенное явление, любой сайт или приложение с интерфейсом создания и отправки сообщений может использоваться для распространения мошеннических, вредоносных или фишинговых сообщений. Классификация текстового содержимого спам-сообщений была приведена в главе 1, поэтому мы не будем подробно рассматривать ее здесь, а главное внимание сосредоточим на методах выявления спамеров на основе их поведения. Вообще говоря, любой сайт, который позволяет пользователям создавать какой-либо контент, может быть использован для генерации спама. Если такой контент представлен в широковещательной форме, а не в формате сообщений, отправляемых от пользователя к пользователю, то другие способы мошенничества или методики оптимизации механизма поиска (search engine optimization – SEO) могут распространять спам в больших масштабах. Атакующий может даже воспользоваться преимуществом так называемых «социальных сигналов»: если популярный контент имеет высокий рейтинг, то поддельные «лайки» или ссылки типа «поделись с друзьями» позволяют спамерам «раскручивать» их сообщения.

Перечисленные выше варианты действий охватывают большинство типов мошенничества в потребительской веб-среде, но этот список далеко не полон. Для получения более подробной информации рекомендуется книга Automated Threat Handbook (**https://www.owasp.org/images/3/33/Automated-threat-handbook.pdf**), изданная организацией Open Web Application Security Project (OWAS), хороший источник, в котором подробно описаны разнообразные типы мошеннических действий.

Типы мошенничества и данные, которые могут защитить

Теперь обсудим более подробно различные способы, которыми атакующие могут попытаться воспользоваться преимуществами потребительских веб-сайтов. В частности, мы будем рассматривать перехват учетной записи, создание учетной записи, финансовое мошенничество и деятельность ботов. Для каждого из этих векторов атаки определим данные, которые необходимо собрать, и сигналы, которые можно использовать для блокирования атаки.

Аутентификация и перехват учетной записи

Любой веб-сайт или приложение, которое занимается только поддержкой контента, не требует каких-либо средств для различения пользователей. Но если на сайте необходимо по-разному оценивать «стаж» и поведение различных пользо-

вателей, например, для того чтобы позволить некоторым пользователям создавать контент или совершать платежи, то потребуется механизм, способный определять, какой именно пользователь выполнил каждый конкретный запрос. А это, в свою очередь, требует определенной формы аутентификации пользователей.

В настоящее время аутентификация в интернете реализована главным образом в форме механизма паролей. Пароли обладают множеством полезных свойств, главным из которых является то, что из-за своей повсеместной распространенности почти каждый пользователь знает, как ими пользоваться. В идеале пароли служат для идентификации пользователя посредством верификации некоторого небольшого фрагмента информации, известной только одному конкретному пользователю. Но на практике пароли перестают быть секретными по нескольким причинам:

- пользователи выбирают пароли, которые легко запомнить, но также легко угадать (наиболее часто выбирается пароль «password», становящийся причиной возникновения большинства уязвимостей);
- поскольку почти невозможно запомнить сложный, единственный в своем роде пароль для каждого сайта, пользователи часто используют одинаковые пароли на нескольких сайтах. Исследования подтверждают, что почти половина пользователей интернета многократно используют одни и те же секретные данные[1]. Это означает, что после взлома одного сайта почти половина его пользователей становится уязвимой и на других сайтах, не связанных со взломанным;
- пользователи уязвимы для фишинговых атак, при которых провокационное сообщение электронной почты или вредоносный веб-сайт принимает вид обычной формы регистрации (входа), чтобы обмануть пользователя и заставить его ввести секретные данные;
- иногда пользователи записывают пароли на бумаге, поэтому их без труда могут узнать коллеги, друзья, члены семьи и т. д., после чего пароли перестают быть секретными.

Что можно предпринять на веб-сайтах для борьбы с этими уязвимостями? Насколько можно быть уверенным в том, что зарегистрировавшееся лицо действительно является настоящим владельцем соответствующей учетной записи?

Одно из направлений исследований ориентировано на разработку альтернативных механизмов аутентификации, которые могут полностью заменить пароли. Биометрические идентификаторы, такие как отпечатки пальцев или снимки радужной оболочки глаза, зарекомендовали себя как надежные идентификаторы, шаблоны поведения, например динамические характеристики ввода с клавиатуры, также могут применяться, но достижения в этой области пока менее значительны. Однако биометрические характеристики по своей природе неизменяемы и неотменяемы. Поскольку существующие в настоящее время сенсорные устройства все-таки можно обмануть, такая уязвимость неприемлема для систем с точки зрения обеспечения надежной защиты даже без учета сложной задачи организации перевода пользователей с привычной, хорошо знакомой парольной системы.

---

[1] *Anupam Das et al*. The Tangled Web of Password Reuse. Proceedings of the 21st Annual Network and Distributed System Security Symposium (2014).

Более общий шаблон – требование второго фактора (second factor) для регистрации (входа) в систему. Такой второй фактор может быть развернут постоянно (например, на сайтах с высокой финансовой ответственностью, таких как банки), подключаться при необходимости (например, для работы с электронной почтой или социальными сайтами (сетями)) или если сам сайт определяет, что попытка регистрации (входа) выглядит подозрительно. Вторые факторы обычно основаны либо на дополнительной информации, известной пользователю, либо используют дополнительную учетную запись или физический предмет, принадлежащий пользователю. Ниже перечислены наиболее часто применяющиеся шаблоны аутентификации с использованием второго фактора:

- подтверждение владения учетной записью электронной почты (с помощью кода или ссылки);
- ввод кода, переданного в смс на мобильный телефон;
- ввод кода, сгенерированного аппаратным ключом (такие ключи предоставляются RSA или Symantec);
- ввод кода, сгенерированного программным приложением (например, Google Authenticator или Microsoft Authenticator);
- ответ на специальный секретный вопрос (например: «В каком месяце родился ваш лучший друг?»);
- верификация социальной информации (например, «фото-CAPTCHA» в Facebook).

Разумеется, все перечисленные вторые факторы имеют свои недостатки:

- учетные записи электронной почты сами по себе защищены паролями, поэтому использование e-mail в качестве второго фактора лишь передает проблему провайдеру e-mail (особенно если электронная почта может использоваться для восстановления пароля для учетной записи);
- коды, передаваемые в СМС, уязвимы для атак типа «человек в середине» или «перенос телефонного номера»;
- аппаратные и программные генераторы кодов безопасны, но пользователь обязательно должен иметь при себе такое устройство или приложение, чтобы пройти регистрацию, кроме того, аппаратное устройство должно быть физически защищено;
- ответы на специальные секретные вопросы могут быть разгаданы;
- социальные подробности чаще всего можно угадать или найти в интернете, или на вопросы не сможет ответить даже пользователь, действительно владеющий учетной записью, – это зависит от того, какая информация была затребована.

В самом экстремальном случае веб-сайт может просто заблокировать учетную запись, если подозревает, что она скомпрометирована, после чего владельцу придется обращаться в службу поддержки клиентов и доказывать свое право владения учетной записью через офлайновые механизмы, например с помощью идентификации по фотографии.

Независимо от используемого механизма восстановления веб-сайт, на котором необходима защита пользователей от перехвата учетных записей, обязательно должен реализовать некоторый механизм, позволяющий отличать легальные случаи регистрации (входа) от незаконных. Это и есть область применения машинного обучения и статистического анализа.

Признаки, используемые для классификации попыток регистрации

С учетом перечисленных недостатков сам по себе пароль недостаточен для надежной проверки идентичности владельца учетной записи. Таким образом, если необходимо защитить учетную запись от посторонних лиц, то при обнаружении правильной комбинации имени пользователя и пароля непременно должна работать некоторая вспомогательная логика, которая принимает решение о выполнении дополнительной процедуры верификации, прежде чем положительно ответить на запрос. Такая логика должна работать как блокирующий механизм и использовать только данные, которые можно собрать в форме регистрации (входа). Кроме того, этот механизм должен пытаться обнаружить все типы атак, перечисленные выше. Из этого следует, что необходимо собирать данные о следующих основных классах сигналов:

- сигналы об атаках грубой силой на одну учетную запись:
  - скорость выполнения попыток входа в учетную запись: часто может оказаться эффективным простой счетчик, ограничивающий количество попыток;
  - рейтинг распространенности паролей, вводимых при попытках: атакующие будут пробовать самые распространенные пароли[1];
  - распределение попыток ввода пароля для конкретной учетной записи: реальные пользователи-владельцы будут пытаться ввести один и тот же пароль или почти одинаковые пароли;
- сигналы, предупреждающие об отклонениях от обычных шаблонов поведения пользователя при попытке регистрации (входа):
  - отличие географического расположения от расположения при предыдущих попытках входа (чем больше удаленность, тем подозрительнее выглядит попытка);
  - использование браузера, приложения, ОС, которые ранее не использовались для этой учетной записи;
  - попытка входа в необычное время суток или в необычный день недели;
  - подозрительно высокая частота попыток входа или слишком малое время между попытками входа;
  - необычная последовательность запросов перед попыткой входа;
- сигналы о крупномасштабной автоматизации попыток входа:
  - большой объем запросов по одному IP-адресу, юзер-агенту или любая другая важная характеристика, извлеченная из данных запросов;
  - чрезмерно большое количество некорректных секретных данных, вводимых при попытках входа на защищаемый сайт;
  - запросы от хостинг-провайдера или с других подозрительных IP-адресов;
  - браузерная телеметрия, свидетельствующая о действиях, которые не выполняются человеком, например оценка скорости нажатия клавиш или цифровые отпечатки устройств.

Конструирование всех этих признаков представляет собой сложную задачу, например для фиксации отклонений от установленных шаблонов потребуется хра-

[1] Поскольку самая надежная практическая методика предусматривает хеширование и «соление» (salt) паролей, на защищаемом сайте, вероятнее всего, ничего не будет известно о применяемых на нем паролях. Рейтинг распространенности можно оценить с помощью опубликованных списков паролей, используемых чаще всего.

нение накопленных данных об этих шаблонах (например, все IP-адреса, которые использовались конкретной учетной записью). Но предположим, что имеется возможность сбора и хранения некоторых или всех этих признаков. А что дальше?

Сначала рассмотрим защиту от атак грубой силой. В большинстве случаев вполне достаточно установить простой ограничивающий счетчик некорректных попыток входа – после достижения некоторого порогового значения можно выполнить блокировку или установить экспоненциально увеличивающиеся интервалы задержки между попытками. Но некоторые сервисы (например, e-mail) могут предлагать приложения, автоматически выполняющие процедуры входа с постоянными интервалами, а определенные события (например, смена пароля) могут привести к тому, что такие автоматизированные попытки входа становятся некорректными, поэтому, вероятнее всего, придется не принимать во внимание попытки, выполняемые с известного устройства.

В зависимости от системных требований к обеспечению безопасности возможность сбора данных о вводимых паролях может отсутствовать. Если есть возможность получения данных о хеш-значениях паролей, введенных за небольшой интервал времени, чтобы подсчитать количество попыток ввода неповторяющихся паролей, то можно установить предельное значение и по этому признаку. Кроме того, может потребоваться блокировка учетной записи, для которой обнаружено чрезмерно большое количество попыток ввода паролей, содержащихся в списках «слабых паролей».

Далее рассмотрим атаки, при которых атакующий уже обладает паролем. Следующий шаг зависит от доступных нам данных с метками.

Присваивание меток типам компрометации учетной записи не является точной наукой. Возможно, у вас есть группа владельцев учетных записей, которые постоянно жалуются на захват их учетных записей посторонними лицами, – такие факты можно пометить как «положительные». Но даже если предположить, что уже существует и действует некоторый механизм защиты (возможно, эвристический), то как можно узнать, какие учетные записи из заблокированных были действительно взломаны и захвачены злоумышленниками? Как отличить их от учетных записей, действительные владельцы которых не смогли (или не пожелали) пройти дополнительную процедуру защиты, созданную для этих учетных записей? А как выделить ложноотрицательные результаты, когда учетные записи захвачены злоумышленниками, но сообщений об этом нет?

Предположим, что в вашем распоряжении нет каких-либо данных с метками. Тогда как можно использовать описанные выше сигналы для определения подозрительных попыток входа?

Для начала можно попробовать методику установки пороговых значений для каждого из перечисленных признаков. Например, может потребоваться проверка таких условий, как регистрация пользователя более 10 раз за один час, или расстояние более 500 миль от любой локации, в которой ранее наблюдался пользователь, или операционная система, которая до этого никогда не использовалась. Каждое из этих пороговых значений должно быть установлено с использованием определенной эвристики, например если только 0,1 % случаев входа пользователей наблюдаются более 10 раз в час, то можно предположить, что это приемлемая пропорциональная доля пользователей, к которым применяются более строгие меры. Но при этом требуется большой объем прогнозирования и интуитивных

предположений, а точная настройка пороговых значений при изменении поведения пользователей является трудной задачей.

Здесь явно необходима научно обоснованная статистическая методика. Отступим на шаг назад и задумаемся о том, что именно необходимо оценить: вероятность того, что лицо, выполняющее попытку входа (регистрации), действительно является владельцем учетной записи. Каким образом можно дать оценку этой вероятности?

Начнем с оценки более простой количественной характеристики, извлекаемой из данных регистрации (входа): вероятность того, что владелец конкретной учетной записи будет входить с определенного IP-адреса, указанного в запросе. Пусть *u* обозначает имя пользователя, о котором идет речь, а *x* обозначает IP-адрес. Простейшая оценка вероятности этого события вычисляется следующим образом:

$$\Pr(\mathrm{IP} = \mathrm{x} \mid \mathrm{User} = \mathrm{u}) = \frac{\#\,\mathrm{logins\ with\ IP} = \mathrm{x\ and\ User} = \mathrm{u}}{\#\,\mathrm{logins\ with\ User} = \mathrm{u}}.$$

Предположим, что у нас имеется пользователь u, для которого зафиксирована следующая хронологическая интернет-статистика:

Таблица 6.1

Дата	IP-адрес	Страна
1 июля	1.2.3.4	US
2 июля	1.2.3.4	US
3 июля	5.6.7.8	US
4 июля	1.2.3.4	US
5 июля	5.6.7.8	US
6 июля	1.2.3.4	US
7 июля	1.2.3.45	US
8 июля	98.76.54.32	FR

По статистическим данным о входах на 6 июля можно вычислить следующую вероятность:

Pr (IP = 1.2.3.4 | User = u) = 0,6.

Но как оценить данные о входах 7 июля? Если вычислять по той же схеме, то в результате получим оценку вероятности, равную нулю. Если вероятность входа пользователя u с IP-адреса *x* равна нулю, то сам факт такого входа обязательно должен быть атакой. Следовательно, если мы применяем такую схему вычисления к номинальному значению, то потребуется считать каждый случай входа с нового IP-адреса подозрительным. В зависимости от типов атак, обнаруженных в вашей системе, и того, какая степень защиты необходима вашей процедуре регистрации (входа), возможно, потребуется подтверждение этого предположения и дополнительная процедура верификации каждого пользователя, изменяющего IP-адреса. (Это будет особенно обременительно для пользователей, которым IP-адреса присваиваются динамически.) Но большинство администраторов предпочтет найти некоторый промежуточный, компромиссный вариант.

Для устранения проблемы нулевых вероятностей можно урегулировать ситуацию, добавив фиктивные («фантомные») события входов в хронологию пользователя. Например, добавляется β событий входов, из которых α выполнены с IP-адреса, о котором идет речь:

$$\Pr(\text{IP} = \text{x} \mid \text{User} = \text{u}) = \frac{(\#\,\text{logins with IP} = \text{x and User} = \text{u}) + \alpha}{(\#\,\text{logins with User} = \text{u}) + \beta}.$$

Точные значения α и β можно подобрать на основе степени подозрительности, ожидаемой от нового IP-адреса, или, выражаясь языком статистики, на основе априорной вероятности. Один из способов оценки априорной вероятности – использование следующих данных: если 20 % всех входов выполнено с IP-адресов, с которых ранее этот пользователь не входил, то вполне разумными значениями будут α = 0,2 и β = 1[1]. С этими значениями случаи входов нашего воображаемого пользователя *u* 6 и 7 июля получаем оценки вероятности соответственно:

Pr (IP = 1.2.3.4 | User = u) = (3 + 0.2)/(5 + 1) = 0,53;
Pr (IP = 1.2.3.45 | User = u) = (0 + 0.2)/(6 + 1) = 0,03.

Отметим, что если рассматривать только IP-адреса, то входы 7 и 8 июля выглядят почти одинаково подозрительными (P = 0,03 в обоих случаях). Но из табл. 6.1 ясно, что вход 8 июля должен быть признан более подозрительным, поскольку выполнен из страны, в которой этот пользователь ранее не находился. Это различие становится еще более очевидным, если выполнить вычисление по странам, аналогичное выполненному для IP-адресов. Установив значения регулирующих факторов α = 0,9 и β = 1, получаем оценки вероятностей для входов 7 и 8 июля соответственно:

Pr (country = US | User = u) = (6 + 0,9)/(6 + 1) = 0,99;
Pr (country = FR | User = u) = (0 + 0,9)/(7 + 1) = 0,13.

Это существенное различие. Подобную методику можно использовать для всех признаков, которым ранее присвоены метки, как «сигналы, сообщающие об отклонениях от ожидаемых шаблонов процедуры регистрации (входа)»: в каждом случае можно вычислить пропорциональную долю входов, совершенных этим пользователем и соответствующих заданному атрибуту, с добавлением соответствующих регулирующих факторов, позволяющих избежать нулевых оценок.

Что можно сказать о признаках «крупномасштабной автоматизации»? Для вычисления скоростных признаков необходимо поддерживать постоянно работающие счетчики запросов за определенный интервал времени, с помощью которых можно учесть каждый входящий запрос. Например, необходимо отслеживать такие характеристики (наряду с другими):

- количество успешных/неудачных попыток входа по всему сайту за последний час/день;

[1] В другой методике регулирования используется свойство принадлежности IP-адресов определенной иерархической структуре (например, IP → ISP → Страна), подробнее см.: *D. Freeman et al*. Who Are You? A Statistical Approach to Measuring User Authenticity. Proceedings of the 23rd Annual Network and Distributed System Security Symposium (2016).

- количество успешных/неудачных попыток входа с каждого конкретного IP-адреса за последний час/день;
- количество успешных/неудачных попыток входа по каждому юзер-агенту за последний час/день.

Чтобы определить аномально большое количество некорректно вводимых секретных данных по всему сайту, можно вычислить коэффициент успешных входов за каждый час в течение достаточно продолжительного интервала времени в прошлом и использовать эти данные для вычисления среднего значения и стандартного отклонения метрики успешных входов. После этого, отслеживая попытки входа, можно вычислять коэффициент успешных входов за каждый прошедший час и определять, достаточно ли близко это значение к хронологически вычисленному среднему значению. Если используются эвристики, то коэффициент неудачных входов с двукратным или трехкратным превышением стандартного отклонения должен активизировать предупреждение или дополнительную функцию. Для получения значения вероятности можно воспользоваться t-тестом для вычисления p-значения. Аналогичные статистические характеристики можно вычислить по коэффициенту неудачных попыток входа по каждому IP-адресу или юзер-агенту.

Наконец, рассмотрим признаки «запросов из подозрительного источника». Для точного захвата этих признаков необходимы репутационные системы (reputation systems), которые присваивают каждому объекту (например, IP-адресу) либо метку, либо числовую оценку, которая обозначает предварительный уровень подозрительности. (Например, провайдеры хостинга, как правило, должны иметь низкую оценку подозрительности.) Создание репутационных систем будет рассматриваться немного позже в этой главе.

Создание собственного классификатора

После того как все перечисленные выше признаки собраны, необходимо объединить их каким-то образом. Без меток эта задача трудновыполнима. Каждый вычисленный признак представляет вероятность, поэтому простейший способ – предположить, что все признаки независимы, и просто перемножить их для получения общей числовой оценки (аналогично можно поступить с данными из журналов и дополнительных источников). Более разумным подходом представляется использование одного из методов выявления аномалий, описанных в главе 3, такого как метод опорных векторов с одним классом, изолирующий лес или метод локального уровня промаха.

Разумеется, с метками можно тренировать классификатор с учителем. Обычно в подобных случаях метки будут чрезвычайно несбалансированными (поскольку очень мало будет данных об атаках, снабженных соответствующими метками), поэтому придется воспользоваться такими методиками, как субдискретизация класса легального трафика (benign) или улучшение функции стоимости для класса атак.

Создание учетной записи

Для похищения средств (активов) пользователя, как правило, необходим захват его учетной записи, но такие атаки, как распространение спама или подделка контента, могут быть выполнены из учетных записей, созданных непосредствен-

но атакующим злоумышленником. Если атакующий имеет возможность создания большого количества учетных записей, то сайт может оказаться переполненным вредоносными пользователями. Таким образом, защита процесса создания учетной записи является важнейшим условием формирования безопасной системы.

Существуют две общие методики выявления и удаления мошеннических учетных записей: числовая оценка запросов на создание учетной записи для полного и абсолютного предотвращения создания мошеннических учетных записей и числовая оценка существующих учетных записей для удаления, блокировки или ограничения действий подозрительных учетных записей. Блокировка в процессе создания лишает атакующего возможности причинить любой вред, а также позволяет сохранять управление размером базы данных учетных записей. Но оценка постфактум дает большую точность и полноту, поскольку сам факт создания учетной записи предоставляет больше доступной информации о ней.

Приведенный ниже (упрощенный) пример демонстрирует применение этого компромисса на практике. Предположим, по имеющимся данным вы сделали вывод о том, что если более 20 новых учетных записей созданы с одного IP-адреса в течение часа, то эти учетные записи определенно являются мошенническими. Если попытаться дать числовую оценку времени создания учетных записей, то можно с помощью счетчика определять количество попыток создания с каждого IP-адреса за последний час и блокировать (вариант: предлагать CAPTCHA или другое средство), когда значение счетчика станет больше 20. (В следующем подразделе более подробно рассматривается организация работы постоянно действующих счетчиков.) Но для каждой конкретной атаки сохраняется возможность создания 20 мошеннических учетных записей, с которых можно без затруднений рассылать спам. С другой стороны, если оценивать количество вновь создаваемых учетных записей только один раз в час и отключать все группы из более чем 20 учетных записей, созданных с одного IP-адреса, то будут заблокированы все спамеры, но у них останется целый час для нанесения ущерба.

Становится понятно, что надежный способ должен объединять элементы обеих методик. В этом разделе рассматривается модель оценки, использующая фактор времени создания учетной записи. Признаки для работы этой модели разделяются на две категории: признаки скорости (velocity features) и оценки репутации (reputation scores).

Признаки скорости

Когда атакующий перегружает систему многочисленными запросами на создание учетных записей, вряд ли необходимо создание сложной модели машинного обучения, – нужно просто заблокировать вредоносный IP-адрес. Но атакующий найдет новый IP-адрес и продолжит свои действия, т. е. начинается бесконечная игра в кошки-мышки. Чтобы прекратить эту игру и создать надежную автоматизированную защиту, используются признаки скорости (velocity features).

Основным компонентом признака скорости является постоянно действующий накопительный счетчик (rolling counter). Постоянно действующий накопительный счетчик делит время на k интервалов (например, каждый час в сутках) и для каждого ключа в некотором пространстве ключей (например, в наборе всех IP-адресов) организует подсчет событий по этому ключу в каждом интервале времени. Через заданные периоды (например, через каждый час) счетчик самого ста-

рого интервала сбрасывается (очищается) и создается счетчик нового интервала. В любой момент можно запросить значение счетчика для получения количества событий для заданного ключа в течение последних t интервалов времени (если $t \leqslant k$).

Но поскольку каждый интервал будет содержать приблизительно одинаковое количество ключей, наличие большого числа чрезмерно детализированных интервалов может привести к резкому росту требований к ресурсам, необходимым для хранения накапливаемых данных (например, при использовании одноминутных счетчиков почасовая оценка будет более точной, чем при использовании десятиминутных счетчиков).

После создания постоянно действующих накопительных счетчиков[1] можно начать подсчет попыток создания учетных записей по любому количеству различных ключей за предварительно определенные интервалы времени. Типы ключей, по которым ведется подсчет, могут быть следующими:

- IP-адрес или IP-подсеть;
- геолокационные признаки (город, страна и т. д.);
- тип или версия браузера;
- операционная система и ее версия;
- успешные/неудачные попытки создания учетной записи;
- признаки пользовательского имени учетной записи (например, подстроки из адреса электронной почты или номер телефона);
- конечный пункт в потоке создания учетной записи (например, мобильный телефон или десктоп);
- API, используемый в конечном пункте;
- любое другое свойство запроса на регистрацию.

Кроме того, можно объединять признаки, например успешные регистрации на десктопе с каждого IP-адреса. Единственным ограничением является ваше воображение и возможности вашей компьютерной системы.

После реализации счетчиков можно приступить к вычислению признаков скорости. Простые скоростные характеристики очевидны: нужно просто извлечь значения счетчиков из определенного количества интервалов времени, просуммировать их, затем разделить на число интервалов (возможно, с преобразованием в необходимые единицы измерения). Если существует общий предельный коэффициент для любого признака, то легко проверить для каждого запроса, не превышает ли текущий коэффициент установленный предел.

Разумеется, применимость общих коэффициентов имеет свои ограничения. Вполне вероятно, что лишь небольшая часть ваших пользователей использует ОС Linux, зато доля пользователей macOS намного больше. Если установлен общий предел на создание учетных записей для каждой ОС в час, то он должен быть установлен так, чтобы соответствовать количеству регистраций из macOS, а максимальное количество учетных записей из Linux становится совсем малозначимым.

Для устранения этой проблемы можно воспользоваться хронологическими данными для определения необычного количества запросов по каждому ключу.

---

[1] Майкл Нолл (Michael Noll) представляет эффективную реализацию алгоритма постоянно действующего накопительного счетчика в своем блоге **https://www.michael-noll.com/blog/2013/01/18/implementing-real-time-trending-topics-in-storm/**.

Сам накопительный счетчик предоставляет некоторые хронологические данные, например если имеются счетчики запросов в час за последние 24 часа, можно вычислить количественную оценку, как показано ниже:

Формула 6.1. Вычисление недавних максимальных значений (пиков)

$$\left(\frac{\#\text{ запросов за последние k часов до настоящего времени}}{\#\text{ запросов в интервале от k часов назад до 24 часов назад} + 1}\right) \Big/ \left(\frac{k}{24}\right).$$

Эта формула дает «величину пикового значения» за последние k часов по отношению к значению за предыдущие 24 – k часов. (Слагаемое + 1 в знаменателе предохраняет от деления на ноль.) Для определения разумного порогового значения необходимо фиксировать в журнале или имитировать значения накопительных счетчиков по хронологическим данным и установить предел, при котором можно без риска блокировать запросы.

Использование недавних (по хронологии) данных для определения пиковых значений применимо только в том случае, когда объем этих данных достаточен. Более надежным средством измерения будет использование хронологических (более ранних) данных, вычисляемых по записям независимых журналов или по счетчикам, функционирующим в течение длительного времени, для установки базиса активных действий для каждого ключа за каждый день (сутки) и последующего использования формулы, аналогичной формуле 6.1, чтобы определить, когда наблюдались пиковые значения. При этом необходимо обязательно обеспечить обработку ненаблюдаемых или редко наблюдаемых ключей. Кроме того, может оказаться полезным регулирование значений счетчиков (методами, перечисленными выше в подразделе «Признаки, используемые для классификации попыток регистрации»).

Выявление пиковых (максимальных) значений является примером практического применения более общего принципа: скоростные характеристики могут быть весьма эффективными, если они объединены в отношения (ratios). Выявление пикового значения по одному ключу использует отношение количества самых последних запросов к количеству более ранних запросов. Но любое из приведенных ниже отношений должно иметь «характерное» («типичное») значение:

- отношение успешных попыток к неудачным попыткам входа или регистрации (максимумы количества неудачных попыток вызывают подозрения);
- отношение API-запросов к запросам страниц (при API-запросе без запроса соответствующей страницы предполагается автоматизация, по крайней мере, когда юзер-агент выдает себя за браузер);
- отношение мобильных запросов к десктопным запросам;
- отношение ошибок (например, коды ответов 4xx) к успешным запросам (с кодом ответов 200).

Непрерывные накопительные счетчики можно использовать для вычисления этих отношений в реальном времени, следовательно, возможно их включение как признаков в классификатор.

Оценки репутации

Скоростные признаки хороши для перехвата крупномасштабных атак, но что, если атакующий замедляет и/или распределяет свои ключи так, чтобы количест-

во запросов всегда было ниже порогового значения для любого произвольно выбранного ключа? Как можно принять правильное решение на основании единственного запроса?

В таких ситуациях основой становится концепция репутации (reputation). Репутация представляет в количественной форме знания, которыми мы располагаем о конкретном ключе (например, об IP-адресе), которые получены из накопленных данных или предоставлены третьими сторонами. Эти знания (данные) образуют исходный пункт (более формально – априорные, или предварительные, данные) для определения оценки того, является ли запрос из рассматриваемого ключа корректным (легитимным).

Рассмотрим пример с IP-адресом. Простейшим репутационным сигналом для модели защиты регистрации является доля учетных записей для конкретного IP-адреса, которые ранее были помечены как некорректные. Более интеллектуальные системы способны захватывать другие метрики некорректности: сколько попыток захвата/передачи учетных записей было зафиксировано с этого IP-адреса или какой объем спама распространяется с этого IP-адреса. Если такие метки недоступны в текущий момент, то можно воспользоваться следующими косвенными признаками для оценки репутации:

- как давно было зафиксировано первое появление этого IP-адреса (чем раньше, тем лучше);
- сколько корректных (легитимных) учетных записей зарегистрировано для этого IP-адреса;
- какой доход получен от этого IP-адреса;
- насколько равномерным и постоянным является трафик с этого IP-адреса (пиковые периоды являются подозрительными);
- каково соотношение операций чтения и записи для этого IP-адреса (больше операций записи = больше спама);
- насколько известным является этот IP-адрес (как часть всех запросов).

В дополнение к динамически собираемым данным немалым подспорьем являются внешние данные, которые могут поступать в репутационную систему, особенно если они касаются IP-адресов и юзер-агентов. Можно подписаться на базу данных, которая будет информировать о том, принадлежит ли любой произвольный IP-адрес провайдеру хостинга, является адресом VPN, или анонимным прокси, или выходным узлом Tor и т. д. Более того, существует множество поставщиков услуг, которые проводят мониторинг больших объемов трафика интернета, формируют и распространяют за плату черные списки IP-адресов. Такие организации обычно предоставляют бесплатные образцы данных, так что можно проверить, насколько точно их метки совпадают с источниками нарушений на вашем сайте.

Эти же организации публикуют списки известных юзер-агентов ботов и пакетов веб-скриптинга. Например, запросы через curl, wget или запросы на языке Python должны классифицироваться как чрезвычайно подозрительные. Аналогично, если запрос представляет себя как Googlebot, то это действительно Googlebot или нечто имитирующее Googlebot в надежде, что ваши потребности в оптимизации механизма поиска позволят им проникнуть на сайт. Но настоящий Googlebot вряд ли будет создавать какие-либо учетные записи.

Теперь рассмотрим, как объединить все эти признаки в оценку репутации, а в качестве примера снова используем IP-адрес. С этой оценкой можно ответить

на вопрос: «Какова вероятность того, что рассматриваемый IP-адрес будет использоваться для нарушений в будущем?» Для интерпретации такого вопроса, как задачи обучения с учителем, необходимы метки. Выбор меток будет зависеть от точного определения того, от чего вы намерены защищаться, но здесь для простоты предположим, что IP-адрес считается «вредоносным», если с него создана мошенническая учетная запись в интервале n дней. Тогда наша задача обучения с учителем формулируется так: рассматривая данные за последние m дней по конкретному IP-адресу, дать прогноз, будет ли обнаружена мошенническая учетная запись, созданная за последующие n дней с того же IP-адреса.

Для тренировки классификатора с учителем вычисляются признаки (начиная с описанных выше) для каждого IP-адреса за период m дней и метки для n дней, следующих непосредственно за исследуемым периодом. Некоторые признаки являются свойствами IP-адреса, например «это провайдер хостинга». Другие основаны на измерениях по времени, например «количество корректных учетных записей для этого IP-адреса». Следует отметить, что можно вычислить количество учетных записей за весь интервал в n дней, или за каждый из n дней, или некоторое усредненное значение. Если объем имеющихся данных достаточен, то можно вычислить n признаков, по одному для каждого дня, чтобы сформировать тренды. Для этого потребуются некоторые эксперименты.

Если планируется регулярная повторная тренировка (например, ежедневная), то, возможно, вполне достаточной окажется валидация на имеющемся тестовом наборе данных (т. е. на произвольном подмножестве данных, исключенных из тренировочного набора). Если повторная тренировка нерегулярна (например, реже, чем каждые n дней), возможно, потребуется убедиться в том, что выбранная модель способна давать достаточно правильные прогнозы, поэтому необходима проверка модели в следующие n дней на тренировочных выборках, как показано на рис. 6.1.

Рис. 6.1 ❖ Нерегулярная тренировка и валидация

Резюмируя все вышесказанное об оценке репутации: в этом разделе в качестве примера в основном использовался IP-адрес. Но все операции применимы к любому из ключей, перечисленных выше, в разделе «Признаки скорости». Единственным ограничением при создании репутационных систем для стран, браузеров, доменов и прочих объектов является объем имеющихся в наличии данных, а также ваши творческие способности.

Финансовое мошенничество

Если атакующие злоумышленники могут получить вашу продукцию без оплаты, то у них появляется возможность ее перепродажи по более низкой цене и получения незаконной прибыли. Эта проблема существует не только для физической

продукции, для которой имеется большой рынок перепродаж (например, электроника), но также для услуг (сервисов), которые могут быть перепроданы на конкурирующем рынке (например, провайдерами мобильной связи на транспорте). Если у вас достаточно обширная клиентура, то всегда найдется несколько человек, которые попытаются похитить вашу продукцию вне зависимости от ее материальной (или нематериальной) сущности. Чтобы остановить таких людей, необходимо точно определять, является ли законной и корректной каждая покупка на вашем сайте.

Подавляющее большинство финансовых мошенничеств совершается с помощью украденных кредитных карт. Мошенничество с кредитными картами появилось раньше интернета, и для исследования и предотвращения этой деятельности была проделана огромная работа. Одна транзакция с помощью кредитной карты при обработке проходит через несколько различных объектов.

1. Продавец (вы).
2. Оператор платежей.
3. Банк продавца.
4. Сеть кредитных карт (Visa/Mastercard/American Express/Discover), которая управляет маршрутизацией транзакций между банками.
5. Банк, выпустивший кредитную карту.

В каждом из этих объектов имеется собственный механизм выявления мошенничества, который может обнаружить мошенническую транзакцию и предъявить обвинение в ее совершении прочим звеньям этой цепочки. Таким образом, цена мошенничества для вас состоит не только в потере конкретной продукции или услуги, но и в отчислениях на каждом уровне этой экосистемы. В экстренных случаях ущерб, нанесенный вашему бизнесу и деловой репутации, может привести к тому, что некоторые банки или платежные сети потребуют выплаты крупных штрафов или даже откажутся иметь дело с вами.

Разумеется, кредитные карты не являются единственным средством оплаты, допустимым на вашем сайте. Прямое дебетование широко распространено в Европе, где кредитные карты труднее получить, чем в США. На многих сайтах разрешено использование онлайновых платежных систем, таких как PayPal, ApplePay или AndroidPay. В подобных случаях вместо данных о кредитной карте и банке вам предоставляются данные об учетной записи и соответствующем онлайновом счете. Но принципы выявления мошенничества, в сущности, остаются одинаковыми для всех типов платежей.

Несмотря на то что многие компании предлагают выявление финансового мошенничества как услугу, вы можете принять решение о создании собственной системы выявления мошенничества. Вероятнее всего, у вас не будет возможности передавать важные секретные данные в стороннюю организацию. Возможно, для вашей продукции существует необычный шаблон выполнения платежа. Или вы рассчитали, что дешевле обойдется создание собственной системы выявления мошенничеств. В любом случае, после принятия такого решения необходимо заняться сбором признаков, характерных для мошеннической операции. Некоторые признаки перечислены ниже:

- профили расходов покупателей:
  - величина стандартного отклонения суммы исследуемой транзакции от средней суммы покупок данного клиента;

  - скорость покупок по кредитной карте;
  - преобладание текущей продукции или категории продукции в хронологии покупок данного клиента;
  - является ли исследуемая транзакция первой покупкой (например, новый пользователь неожиданно начинает совершать большое количество покупок);
  - является ли этот способ оплаты/тип карты обычным для данного клиента;
  - давно ли этот способ оплаты был добавлен в учетную запись клиента;
- взаимосвязи географических/временны́х факторов:
  - все взаимосвязанные (корреляционные) сигналы, необходимые для аутентификации (например, географическое местоположение и перемещение, хронология использования IP-адреса/браузера);
  - скорость географических перемещений (например, если транзакция с использованием материальной кредитной карты была совершена в Лондоне в 8:45 вечера, затем в Нью-Йорке в 9:00 вечера в тот же день, то скорость перемещения пользователя аномально высока);
- несоответствие в данных:
  - соответствует ли адрес операции с кредитной картой информации в профиле пользователя на уровне города/штата (области, провинции)/страны;
  - несоответствие между адресами оплаты и доставки;
  - находится ли банк, выпустивший кредитную карту, в той же стране, что и пользователь;
- профиль учетной записи:
  - возраст учетной записи данного пользователя;
  - оценка репутации создаваемой учетной записи (*см*. предыдущий раздел);
- статистические данные о взаимодействии с покупателем:
  - сколько раз этот покупатель проходил полную процедуру (поток) платежа;
  - количество используемых кредитных карт (попыток использования);
  - сколько раз использовалась (или совершалась попытка использования) данная карта;
  - количество заказов по каждому счету или по каждому адресу доставки.

Если необходима тренировка алгоритма выявления мошенничества с учителем или, проще говоря, если необходимо знать, насколько успешно выполняется перехват мошеннических действий, то потребуются данные с метками. «Золотым стандартом» для данных с метками являются возвратные платежи – покупки, которые реальный владелец карты объявил мошенническими, а банк аннулировал соответствующий платеж и вернул деньги. Но, как правило, выполнение возвратного платежа требует не менее месяца, а во многих случаях этот срок может затянуться до шести месяцев. Поэтому возвратные платежи невозможно использовать для динамических метрик или для исследования того, как противник адаптируется к недавним изменениям. Таким образом, если необходимы быстрые и многократно повторяемые измерения для моделей выявления мошенничества, то потребуется вспомогательная метрика, любая из перечисленных ниже:

- сообщение покупателя о мошенничестве;
- возвраты оплаты покупателем;
- покупки, сделанные из мошеннических или скомпрометированных учетных записей.

Наконец, при создании собственной системы важно правильно определить в платежном процессе (потоке) пункт, в котором будет интегрирована система выявления мошенничества. Как и при борьбе с другими типами нарушений, чем длиннее подготовительный период, тем больше данных можно собрать для принятия более обоснованного решения, но вместе с тем возрастает и опасность причинения большего ущерба. Ниже перечислены некоторые возможные пункты внедрения системы выявления мошенничества:

- перед авторизацией: возможно, потребуется определение хотя бы минимальной оценки перед отправкой каких-либо данных в компанию, выпустившую кредитную карту. Если отвергнуто слишком много карт, могут последовать штрафные санкции. Более того, проверки перед авторизацией не дают атакующим возможности использовать ваш сайт как тестовый стенд для определения работоспособности похищенных кредитных карт;
- после авторизации, перед покупкой – это обычное место функционирования системы оценки мошенничества, так как позволяет блокировать карты, отвергнутые банком, и избежать процедур возвратных платежей, т. е. вы не будете субсидировать мошеннические покупки. Если на сайте правильно настроены функции аутентификации/перехвата нарушителей, то можно позволить покупателю продолжить процесс оформления покупки, как если бы он был корректным, и собрать больше данных для более точной оценки, но завершить этот процесс до момента получения реальных денежных средств;
- после покупки: если ваша продукция материальна и требует некоторого времени для подготовки к отправке ее покупателю или если вы предлагаете виртуальный сервис (услугу), который не способен причинить большого ущерба за короткий период непосредственно после продажи, можно позволить провести процесс покупки и оплаты до конца и собрать больше поведенческих сигналов перед принятием решения по покупке и отмене/возмещению сумм транзакций, которые признаны мошенническими. Но при оценке после продажи вам придется выполнять процедуры возвратных платежей в тех случаях, когда реальный владелец быстро выявляет мошенничество.

Деятельность ботов

В некоторых случаях атакующие злоумышленники могут сосредоточиться на изъятии ценностей у одной жертвы. Самым очевидным примером является банковский счет, но любая учетная запись, так или иначе связанная с хранением любых ценных активов, может стать целью. Это могут быть учетные записи для совместной аренды транспортных средств и жилья, премиальные или бонусные счета, рекламные фонды и т. д. Для таких учетных записей, связанных с реальными ценностями, атакующие оправданно проделывают большой объем ручной работы, чтобы избежать обнаружения. С другой стороны, во многих случаях ожидаемая выгода от атаки на одну жертву очень мала, часто она не превышает стоимость человеческих трудозатрат, требуемых для получения доступа к учетной записи

и ее использования. Примерами могут служить рассылка спама, подделка удостоверений личности и искажение данных. В этих случаях атакующие непременно должны использовать автоматизацию, если надеются получить хоть какую-то прибыль. Даже в ситуациях, сулящих большую выгоду, человеческие трудозатраты со временем только увеличиваются, поэтому автоматизация с большой вероятностью обеспечивает более надежную компенсацию издержек для атакующей стороны.

Из этого следует, что обнаружение нарушений в большинстве случаев равнозначно обнаружению автоматизированной деятельности (т. е. ботов) на сайте или в приложении[1]. Боты могут пытаться предпринимать любые типы действий, включая следующие:

- создание учетной записи: эта тема подробно рассматривалась в предыдущем разделе. Ботов этого типа можно заблокировать до или после создания учетной записи;
- подделка данных, удостоверяющих личность: при этом используются полученные каким-либо способом списки пар имя пользователя/пароль для атаки на инфраструктуру входа/регистрации на вашем сайте в попытках взлома учетных записей. Боты этого типа должны быть заблокированы, прежде чем получат доступ к учетной записи, в идеальном варианте – без утечки информации о правильности (или неправильности) учетных данных;
- кража (данных) – загрузка данных с вашего сайта для перепродажи или для других незаконных действий. Боты-воры должны быть заблокированы до того, как они доберутся до каких-либо данных, поэтому обнаружение обязательно должно быть синхронизировано и выполняться очень быстро, для того чтобы не создавать слишком больших задержек для добропорядочных пользователей;
- кликфрод – накрутка счетчиков кликов для получения дополнительного дохода на сайте поддержки рекламы или использование искусственных кликов для исчерпания рекламного бюджета конкурентов. Минимальное требование – рекламодателям не должны засчитываться мошеннические (искусственные) клики. Вычисление/выявление должно выполняться в реальном времени или может быть отложенным – ежемесячно или в ежеквартальном зачетном цикле. Но если самые свежие данные о клике используются для выявления текущего кликфрода для рекламных накруток, то мошеннические клики должны обрабатываться практически в реальном времени (хотя и асинхронно);
- рейтинговое мошенничество – искусственное увеличение счетчиков просмотров (посещений), лайков или ссылок типа «поделись с друзьями», для того чтобы распространить контент для более широкой аудитории. Подобные мошеннические действия должны быть пресечены как можно быстрее после их обнаружения;
- онлайновые игры: боты могут имитировать деятельность, которая утомительна или требует больших человеческих трудозатрат, например переме-

[1] Такая равнозначность исчезает, если вы разрешаете или даже поддерживаете автоматизацию действий, например если у вас функционируют API, которые могут регулярно проводить (автоматические) опросы с целью получения данных.

щение на большие (по географическим меркам) расстояния (с помощью поддельных сигналов GPS), или зарабатывать (бонусные) баллы и другие игровые валюты с помощью многократно повторяемых действий (например, многократно повторяющееся сражение с одним и тем же врагом для получения большого числа очков опыта).

Как и в сфере обнаружения финансовых мошенничеств, существуют компании, которые специализируются исключительно на выявлении и блокировании деятельности ботов. Здесь мы приводим несколько основных рекомендаций по выявлению и блокированию атак ботов.

Боты могут обладать совершенно различными уровнями интеллекта и иметь самые разнообразные намерения. Многие боты даже являются вполне законными, например поисковые роботы Googlebot или Bingbot. Такие боты обычно сами заявляют о себе и своих целях и заслуживают занесения в файл *robots.txt*, размещаемый на сайте для указания путей, ограничивающих действия ботов.

Самые примитивные боты сообщают все о себе в строке определения юзер-агента. К этой группе относятся инструментальные утилиты curl и wget, программные механизмы, подобные python-requests, или скрипты, выдающие себя за легитимных поисковых роботов, таких как Googlebot. (Последний случай можно распознать по тому факту, что исходным пунктом бота не является корректный IP-адрес поискового механизма.)

После нейтрализации «глупых» ботов весьма важным этапом определения другой автоматизированной деятельности является агрегация (aggregation), при которой подразумевается возможность группирования запросов, приходящих от одного объекта. Если от пользователей обязательно требуется вход (регистрация) на сайт для выполнения действий, которые боты пытаются автоматизировать, то в вашем распоряжении уже имеется идентификатор пользователя, по которому можно проводить агрегацию. Если идентификатор пользователя недоступен, или необходима агрегация по нескольким пользователям, можно взять за основу один или несколько IP-адресов, источники ссылок, юзер-агенты, идентификаторы мобильных приложений или другие характеристики (измерения).

Теперь предположим, что уже агрегированы запросы, которые, как вы полагаете, исходят от одного объекта. Как определить, действительно ли эта деятельность является автоматизированной? Здесь основная идея состоит в том, что шаблон (pattern) запросов от ботов отличается от шаблонов действий, производимых людьми. Ниже перечислены характерные сигналы, для которых возможна числовая оценка:

- скорость запросов: боты выполняют запросы гораздо быстрее, чем люди;
- регулярность запросов, измеряемая как различие интервалов времени между запросами: боты выполняют запросы с более регулярными (практически постоянными) интервалами времени по сравнению с людьми. Даже если оператор бота ввел некоторый фактор случайности для времени запроса, распределение интервалов времени между поступлением запросов остается отличающимся от аналогичной характеристики для запросов, выполняемых людьми;
- хаотичность выбора запрашиваемых путей/страниц: боты сосредоточены на своих конкретных целях, а не на просмотре различных частей сайта. Боты, ворующие данные, будут запрашивать каждую страницу только одно-

кратно, в то время как реальные пользователи многократно посещают наиболее популярные страницы;

- повторяющиеся шаблоны в запросах: бот может многократно запрашивать последовательность страниц A, B, C, чтобы автоматизировать поток выполнения;
- необычные переходы: например, бот может сразу перейти в конечный пункт генерации контента без загрузки страницы, содержащей форму подтверждения;
- различия в заголовках: боты могут циклически менять IP-адреса, юзер-агенты, источники ссылок и прочие элементы заголовков клиентской стороны, чтобы выглядеть как люди, но распределение, генерируемое ботом, вряд ли будет соответствовать типичному распределению на вашем сайте. Например, пользователь-бот может выполнить абсолютно одинаковое количество запросов от каждого из нескольких юзер-агентов;
- различия в закладках (cookies): обычный веб-сайт устанавливает закладки сеанса и может устанавливать другие закладки в зависимости от выполняющегося потока. Боты могут игнорировать некоторые или все запросы на установку закладок, а это приводит к необычным различиям в закладках;
- распределение кодов ответов: большое количество ошибок, особенно с кодами 403 или 404, может указывать на бота, скрипты которого ориентированы на более старую версию сайта.

Различные системы выявления ботов, описанные в литературе, используют некоторые или все перечисленные выше сигналы. Например, система PubCrawl[1] применяет кластеризацию и анализ временны́х рядов для обнаружения распределенных поисковых роботов, а алгоритм Ванга (Wang) и др.[2] выполняет кластеризацию последовательностей запросов на основе метрики схожести.

Иногда невозможно агрегировать запросы по идентификатору пользователя или любому другому ключу, т. е. отсутствует надежный способ определения факта отправки любых двух запросов одним источником. Например, это происходит, если защищаемый конечный пункт открыт для всеобщего доступа без процедуры регистрации (входа) или если теряется возможность управления множеством различных учетных записей во время атаки. В таких случаях необходимо использовать систему выявления ботов на основе запросов (request-based), т. е. определять, является ли запрос автоматизированным, только с помощью данных, содержащихся в самом запросе, без каких-либо внешних счетчиков или временны́х рядов.

Полезные данные, которые можно извлечь из запроса, перечислены ниже:

- способность клиента выполнять код JavaScript. Существуют различные способы получения возможности выполнения кода JavaScript: от простых переадресаций до сложных потоков типа вызов–ответ; они различаются по вводимому интервалу задержки, а также по сложности ботов, которую можно обнаружить;

---

[1] *Grégoire Jacob et al.* PUBCRAWL: Protecting Users and Businesses from CRAWLers. Proceedings of the 21st USENIX Security Symposium (2012).

[2] *Gang Wang et al*. You Are How You Click: Clickstream Analysis for Sybil Detection. Proceedings of the 22nd USENIX Security Symposium (2013): 241–255.

- следы элемента HTML5 canvas, по которым можно определить, является ли поддельным юзер-агент[1];
- порядок, регистр букв и орфография заголовков запросов, которые можно сравнить с корректными запросами от декларируемого юзер-агента;
- цифровые отпечатки TLS, которые можно использовать для идентификации конкретных клиентов;
- орфография и регистр букв значений в полях запросов HTTP с конечным числом возможных вариантов (например, `Accept-Encoding`, `Content-Type`) – скрипты могут содержать опечатки или использовать нестандартные значения;
- данные мобильной аппаратуры (например, микрофон, акселерометр), которые можно использовать для подтверждения того факта, что декларируемый мобильный запрос действительно выполнен с реального мобильного устройства;
- информация об IP-адресе и браузере/устройстве, которую можно использовать для поиска ранее определенных оценок репутации.

При реализации системы выявления на основе запросов неизбежен обычный компромисс между действиями по предотвращению нарушения и затруднениями для добропорядочных пользователей. Если на каждый запрос выставляется интерактивная CAPTCHA, то будет остановлено подавляющее большинство ботов, но в то же время многие ваши добропорядочные пользователи будут недовольны и даже могут отказаться от ваших услуг. Менее экстремальный пример: выполнение кода JavaScript для сбора информации о браузере или других данных может привести к неприемлемым задержкам при загрузке страницы. В конечном итоге только вы можете правильно определить отношение своих пользователей к используемым процессам сбора данных.

Метки и метрики

Присваивание меток запросам ботов для вычисления метрик или для тренировки моделей с учителем представляет собой трудную задачу. В отличие от задачи классификации спама, которую можно передать для выполнения людям, не существует надежного способа, воспользовавшись которым, рецензент мог бы с уверенностью назначить метку «бот» или «не бот» любому отдельно взятому запросу. Поэтому необходимо рассмотреть некоторые альтернативные решения.

Первый набор методов для ботов, который рекомендуется использовать, – боты, объявляющие себя «как есть» в своем заголовке User-Agent. Доступны списки, предоставляемые сообществом ПО с открытым исходным кодом[2], и проприетарные списки, а также характеристики этих ботов, которые остаются применимыми и для более «интеллектуальных» ботов.

Если в вашем случае боты ориентированы на автоматические операции записи (например, спам, ссылки типа «поделись с друзьями», клики), то, вероятнее всего, вы уже получали жалобы и располагаете зафиксированными данными, сгенерированными ботами. Исходный набор данных может стать хорошим отправным

---

[1] *Elie Bursztein et al*. Picasso: Lightweight Device Class Fingerprinting for Web Clients. Proceedings of the 6th Workshop on Security and Privacy in Smartphones and Mobile Devices (2016): 93–102.

[2] Например, в репозитории crawler-user-agents (**https://github.com/monperrus/crawler-user-agents**).

пунктом для тренировки моделей или для кластеризации. Кроме того, можно исследовать мошеннические учетные записи с достаточно большим количеством операций записи.

Если в вашем случае боты ориентированы на автоматические операции чтения (например, похищение данных), т. е. возможность идентифицировать конкретные случаи с нанесением значительного материального ущерба, например с определенного IP-адреса или юзер-агента в определенный день. Пока вы собираете признаки, необходимые для идентификации конкретного события, остальными данными можно воспользоваться для тренировки моделей.

Упомянутый последним подход применяется более широко: вы никогда не должны измерять деятельность ботов, используя тот же сигнал, который применяется для предотвращения этой деятельности. Простейший пример: если выявлены и учтены как нарушители все самоидентифицирующиеся боты с последующей их блокировкой, то в действительности проблему ботов вряд ли можно считать решенной окончательно, даже если выбранные метрики ботов стремятся к нулю. Вы всего лишь стимулировали ботов становиться более интеллектуальными.

С другой стороны, процесс (поток) измерения может быть настолько сложным, насколько вам необходимо, поскольку он не обязан соответствовать требованиям эксплуатационной среды. Таким образом, можно вычислять дополнительные агрегации или обрабатывать асинхронные сигналы браузера, которые, возможно, связаны с существенными издержками при реализации в механизме онлайновой защиты, и использовать их для количественной оценки прогресса в борьбе с ботами. По крайней мере, можно тренировать модель с учителем для процесса измерения, пока собираемые признаки не коррелированы с признаками, используемыми для модели выявления.

Заключительное замечание: возможно, что на практике вы не будете иметь дело с реальной автоматизированной деятельностью ботов, но все же придется уделять внимание количеству обращений к страницам и другим генерируемым статистическим характеристикам. Это общеизвестная проблема чистоты/загрязнения метрик (metrics pollution): так как большую часть всего трафика интернета составляют боты, высока вероятность того, что используемые рабочие метрики будут совершенно другими, если в процесс их вычисления включены боты. Проблему чистоты метрик можно решить, используя те же методы, что и при онлайновом выявлении ботов, в частности применяя методы оценки на основе агрегации и/или на основе данных запросов. Это даже лучше, что нет необходимости реализации решения в эксплуатационной среде реального времени, потому что результатом работы такой системы выявления ботов является только изменение отслеживаемых метрик. Следовательно, требования к интервалам задержек гораздо более мягкие или вообще отсутствуют (например, при использовании Hadoop), и вы можете применять больше данных и/или выполнять более ресурсозатратные вычисления без риска для конечного результата.

Обучение с учителем для решения задач по выявлению нарушений

После завершения вычислений признаков скорости и оценок репутации по многим различным ключам вы готовы к созданию классификатора операций

создания учетных записей. Сначала необходимо определить набор легальных и мошеннических запросов на создание учетных записей с соответствующими метками, затем вычислить признаки для каждого запроса в наборе, выделить тренировочный, тестовый и валидационный наборы и применить любой из алгоритмов обучения с учителем, описанных в главе 2. Все это выглядит не слишком сложным.

Но не следует торопиться. Существуют некоторые тонкости, которым необходимо уделить особое внимание, чтобы обеспечить высокую эффективность. В рассматриваемом ниже примере используется классификатор операций создания учетных записей, но все изучаемые здесь приемы и решения применимы к любому классификатору, создаваемому с использованием состязательных данных.

Метки для данных

В идеальном случае вы имеете большой набор данных, специально подобранных для этого конкретного проекта и снабженных метками, присвоенными вручную. Если это действительно так, то вам очень повезло. Но в реальной жизни данные с метками, аккуратно присвоенными вручную, встречаются чрезвычайно редко. Даже если удается получить некоторое количество таких данных, их может оказаться недостаточно для тренировки надежно работающего классификатора.

Противоположная ситуация: допустим, что у вас вообще нет возможности сделать какую-либо выборку данных и присвоить им метки. В этом случае обучение с учителем бесполезно? Вероятнее всего, нет – ведь у вас непременно должны существовать некоторые учетные записи, которые уже заблокированы на сайте за мошенничество или спам. Если даже таких учетных записей нет, то, возможно, проблема не настолько велика, чтобы обосновать конструирование крупномасштабного классификатора на основе методов машинного обучения. Ранее заблокированные учетные записи могут быть уже распределены по категориям с помощью системы, основанной на правилах, других моделей машинного обучения или ручного вмешательства по конкретным жалобам пользователей. В любом случае, можно использовать эти учетные записи как положительные (т. е. мошеннические) экземпляры[1], а все незаблокированные (пока еще) учетные записи как отрицательные экземпляры.

Теперь рассмотрим риски, связанные с применением этой методики. Первый риск – потеря видимости: модель обучается только тому, что уже известно об учетных записях на вашем сайте. Без назначения меток вручную повторная тренировка не поможет модели идентифицировать новые типы нарушений, поэтому потребуется создание еще одной модели или добавление правил, чтобы справляться с новыми типами атак.

Другой риск – зацикливание обратной связи: модель обучается сама по себе и преувеличивает ошибки. Предположим, что вы ошибочно заблокировали несколько учетных записей из Лихтенштейна. После этого модель в процессе обучения узнает, что учетные записи из Лихтенштейна предопределены как наруши-

[1] Здесь и далее мы будем называть мошеннические запросы «положительными», а корректные запросы – «отрицательными». Выбор конкретных меток – 0 или 1 – для мошеннических запросов мы оставляем на ваше усмотрение. Но при этом будьте последовательны.

тели, и блокирует их с соответствующей частотой. Если повторные тренировки модели проводятся регулярно и ложноположительные случаи не скорректированы, то такое зацикливание обратной связи в конце концов может привести к блокировке всех учетных записей из Лихтенштейна.

Чтобы как-то исправить ситуацию при использовании неправильно размеченных данных, можно принять следующие меры:

- выборка с запасом экземпляров с метками, назначенными вручную, в тренировочных данных;
- выборка с запасом ложноположительных экземпляров в этой модели при повторной тренировке (как при усилении или бустинге (boosting));
- субдискретизация положительных экземпляров из предыдущих итераций этой модели (или тесно связанных с ней моделей/правил);
- если имеются ресурсы для выборки и ручного назначения меток некоторым учетным записям, то рекомендуется использовать их для уточнения спорных случаев вблизи границы решения работающей модели (как при активном обучении[1]).

Последнее предупреждение при использовании тренировки на ненадежных данных: не воспринимайте слишком буквально числовые значения точности и полноты, получаемые на этапе валидации (проверки). Если модель выполняет осмысленную работу по обобщению имеющихся данных, то она найдет ошибки в наборе данных с метками. Это будут «ложноположительные» и «ложноотрицательные» результаты по отношению к ранее назначенным меткам, но в действительности это объекты, скорректированные вашим классификатором. Следовательно, вы должны ожидать более высокой точности при развертывании модели по сравнению с точностью, полученной в результате офлайновых экспериментов. Числовая оценка этого эффекта может быть определена только с помощью онлайновых экспериментов.

Холодный запуск и горячий запуск

При отсутствии оценок операций создания учетных записей («холодный запуск») методика тренировки модели вполне ясна: использование любых меток, которые имеются при создании модели. С другой стороны, если уже существует модель, работающая во время создания учетных записей и блокирующая некоторые запросы («горячий запуск») , то только вновь создаваемые некорректные учетные записи действительно прогоняются через существующую модель (т. е. ложноположительные объекты). Если вы тренируете вторую версию модели только на этих данных, то вторая версия может «забыть» характеристики первой версии.

Чтобы показать сущность этой головоломки на примере, предположим, что версия 1 блокирует все запросы с IP-адреса, если скорость их выполнения больше пяти в день. В дальнейшем характеристика скорости «плохих» учетных записей, используемых для тренировки модели версии 2, становится очень низкой, и старая характеристика, возможно, становится не столь значимой для второй версии модели. В результате при развертывании версии 2 возникает риск пропуска (т. е. разрешения) высокоскоростных запросов с одного IP-адреса.

---

[1] Более подробно см. *Burr Settles*. Active Learning Literature Survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison (2010).

Для устранения этой проблемы существует несколько различных подходов:

- никогда не отбрасывать тренировочные данные – необходимо продолжать сбор данных для модели версии 1 даже после развертывания версии 2 и проводить тренировку на объединенном наборе тренировочных данных для версий 1 и 2 (если наблюдаются быстро изменяющиеся противники, то следует рассмотреть возможность применения модели, действующей по закону радиоактивного распада, чтобы самые последние атаки имели больший вес);
- использовать модели одновременно – тренировать и развертывать версию 2 модели, но одновременно оставить в работе и версию 1. Наиболее вероятной причиной создания версии 2 является снижение эффективности версии 1. В этом случае необходимо точно настроить пороговое значение для версии 1, чтобы сделать ее более точной. Такой подход может привести к резкому росту сложности, по мере того как развертывается все больше новых моделей;
- выборка положительных результатов из развернутой (работающей) версии 1 модели для расширения набора тренировочных данных для версии 2 – преимущество этого подхода состоит в том, что все тренировочные данные являются актуальными. Но имеется и недостаток: трудно определить ложноположительные результаты в версии 1 (*см*. следующий раздел).

Ложноположительные и ложноотрицательные результаты

Теоретически ложноотрицательные результаты легко идентифицировать – это учетные записи, которые модель определила как корректные, и им была разрешена регистрация, но в итоге они оказались некорректными. Однако на практике идентифицировать ложноотрицательные результаты в работающей модели можно, только если они инициируют работу некоторой другой модели или правила либо приводят к жалобам пользователей. Всегда существуют некорректные учетные записи, которые остались незамеченными, следовательно, им присвоена неправильная метка. При наличии ресурсов для выборки и назначении правильных меток экземплярам появляется возможность наиболее эффективно выполнять выборку учетных записей с оценками, близкими к пороговому значению классификации. Это экземпляры, вызывающие у модели наибольшие сомнения.

Идентификация ложноположительных результатов не менее трудна. Они блокируются с использованием наилучшей информации, которой вы располагаете в текущий момент, следовательно, вы не получаете дополнительной информации, которая позволит улучшить ваше решение. Еще хуже то, что ложноположительный результат при создании учетной записи означает, что вы заблокировали нового пользователя, который пока еще даже не определил взаимоотношения с вашим сайтом. Такой пользователь скорее откажется от ваших услуг, чем подаст жалобу в группу сопровождения.

Один из способов решения проблемы идентификации ложноположительных результатов – позволить небольшой части запросов, определенных как некорректные, все-таки создавать учетные записи и постоянно наблюдать за ними, чтобы в дальнейшем выявить их вредоносную деятельность. Такая методика неэф-

фективна, поскольку атакующий, у которого прошло всего лишь пять процентов попыток регистрации, может просто отказаться от распространения спама, так как нет достаточного масштаба, – пробный подход привел к тому, что противник изменил свое поведение, следовательно, результаты такого подхода, вероятнее всего, не позволят сделать правильный вывод. (Эта проблема существует при A/B-тестировании состязательных моделей в целом.) К тому же вы должны постоянно наблюдать за легальной деятельностью таких учетных записей, чтобы убедиться в том, что это «действительно ложноотрицательные результаты».

Выявленные и правильно идентифицированные ложноположительные и ложноотрицательные результаты могут быть введены как выборка с запасом в тренировочные данные для повторных тренировок модели с использованием одной из методик, описанных в предыдущем разделе.

Несколько вариантов ответной реакции

Обычно модель оценки операции создания учетной записи имеет несколько пороговых значений: блокировка, если оценка чрезвычайно плоха, разрешение выполнения запроса, если оценка достаточно хороша, или выполнение дополнительной проверки (например, CAPTCHA или подтверждение по телефону) для оценок из среднего диапазона. Такой подход может усложнить измерение эффективности и назначение меток для повторной тренировки: является ли некорректная учетная запись, которая прошла дополнительную проверку, истинно положительным или ложноотрицательным результатом? Что можно сказать о корректной учетной записи, которой пришлось проходить дополнительную проверку? В идеале каждый подобный случай может получить конкретную стоимость в используемой глобальной функции стоимости. Если вы не используете функцию стоимости, то должны сами принять решение о том, что считать положительными и отрицательными результатами.

Крупномасштабные атаки

И при холодном запуске, и при использовании методики «выборки положительных результатов» для устранения проблемы горячего запуска может возникать следующая ситуация: сначала появился один не слишком хитроумный атакующий злоумышленник, который выполнил так много запросов с одного IP-адреса, что его авторство было определено для половины некорректных запросов в наборе тренировочных данных. Если просто начать тренировку с таким распределением, то ваша модель обучится тому, что половина всех атак выглядит как атака этого злоумышленника, – это типичный случай переподгонки по единственному конкретному событию просто потому, что атака была настолько крупномасштабной (и в случае горячего запуска она была даже отражена).

Одним из способов устранения такой проблемы является субдискретизация крупномасштабных атак. Например, из атаки с x запросами можно сделать выборку log(x) запросов для включения в тренировочный набор данных. После этого подобная атака будет выглядеть большой, но не огромной.

При таком подходе возникает вопрос: как идентифицировать атаки от одного действующего лица? Ответ на этот вопрос будет дан в следующем разделе.

Кластеризация нарушений

Захват одной учетной записи может быть чрезвычайно ущербным для конкретной жертвы, но единственная мошенническая учетная запись вряд ли станет причиной всеобщего хаоса, особенно если объем деятельности одной учетной записи можно ограничить[1]. Таким образом, для расширения масштаба своего мошенничества атакующие вынуждены создавать много учетных записей. Аналогично, поскольку ожидаемая выгода от единичных спам-сообщений невелика, злоумышленники должны рассылать тысячи и даже миллионы сообщений, чтобы получить приемлемую компенсацию. Подобное обоснование можно применить практически для любого вида мошенничества: оно эффективно работает, только если атакующий может выполнить большой объем мошеннических действий за достаточно короткий интервал времени.

Таким образом, мошенническая деятельность на сайте отличается от корректных действий главным образом в том, что неправомерные действия координируются между несколькими (многими) учетными записями. Более опытные мошенники попытаются маскировать свой трафик, выдавая его за корректный с помощью изменения свойств запроса, например направляя запросы с различных IP-адресов, разбросанных по всему миру. Но даже такое изменение свойств имеет свои пределы – почти всегда какое-то свойство или несколько свойств мошеннических запросов оказываются «очень похожими» друг на друга.

Алгоритмическим подходом к реализации этого интуитивного решения является кластеризация (clustering): определение групп объектов, похожих друг на друга в некотором математическом смысле. Но точного разделения учетных записей или событий на группы недостаточно для выявления мошеннической деятельности – необходимо также определить, является ли каждый кластер корректным или некорректным. Наконец, нужно исследовать кластеры нарушителей с целью поиска ложноположительных результатов, т. е. учетных записей, которые случайно оказались в группе нарушителей.

В итоге процесс кластеризации можно определить следующим образом:

1) группирование учетных записей или действий по кластерам;
2) определение для каждого кластера как единого целого, является он корректным или некорректным;
3) внутри каждого некорректного кластера выполняется поиск и исключение всех корректных учетных записей или действий.

Для этапа 1 существует множество доступных методов кластеризации. Некоторые методы мы кратко рассмотрим немного позже. Этап 2 – это этап классификации, следовательно, можно применить методы обучения с учителем или без учителя, в зависимости от наличия данных с метками. Здесь мы не рассматриваем подробно этап 3. Один из вариантов решения состоит в рекурсивном повторении этапов 1 и 2.

---

[1] Тут необходимо применять действенное правило: «ресурсы не должны быть бесконечными». На практике это означает, что должны быть установлены глобальные ограничители всех типов деятельности для каждой учетной записи – по количеству попыток регистрации (входа), сообщений, транзакций и даже по количеству обращений к страницам.

Следует отметить, что необходимо правильно выбрать значения двух важных параметров, которые в высшей степени зависимы от предметной области:

- размер кластера, требуемый для признания его значимым. Большинство корректных действий и некоторые мошеннические действия будут не согласованными между собой, поэтому потребуется удаление данных, которые не подлежат кластеризации в достаточно большую группу;
- степень некорректности кластера, необходимая для пометки его как мошеннического. Это более значимо для варианта обучения с учителем, при котором алгоритм будет обучаться на кластерах как целостных объектах. В некоторых случаях единственного «плохого» объекта в кластере достаточно, чтобы «испортить» весь кластер. Примером являются фотографии в профиле в социальной сети: почти любая учетная запись, делящаяся фотографиями с некорректной учетной записью, также становится некорректной. В других случаях потребуется, чтобы подавляющее большинство действий в кластере было некорректным. Пример: группы IP-адресов, когда необходима полная уверенность в том, что абсолютное большинство IP-адресов обслуживает вредоносный трафик, прежде чем пометить весь кластер как мошеннический.

Пример: кластеризация доменов спама

Чтобы продемонстрировать практическое применение методов генерации кластеров и оценки (классификации) кластеров, мы будем работать с набором данных с метками, содержащим имена доменов интернета. Корректные имена взяты из источника «Top 500,000 Alexa sites form May 2014» (**https://hackertarget.com/500k-http-headers/**), а некорректные имена представлены 13 788 «зараженными доменами» с сайта stopforumspam.org (**http://www.stopforumspam.com/downloads**). Доля позитивных экземпляров (т. е. источников спама) в наборе данных составляет 2,7 %. Это вполне разумное соотношение с точки зрения порядка величины для типичных практических задач по выявлению мошеннической деятельности.

Данные о доменах – это не данные об учетных записях или операциях, но они обладают свойством, которое можно использовать для определения кластеров разумного размера. Например, при быстром просмотре некорректных доменов в алфавитном порядке обнаруживаются следующие кластеры с размером, не меньшим 10:

```
aewh.info, aewn.info, aewy.info, aexa.info, aexd.info, aexf.info, aexg.info, aexw.info, aexy.
info, aeyq.info, aezl.info

airjordanoutletcenter.us, airjordanoutletclub.us, airjordanoutletdesign.us,
airjordanoutletgroup.us, airjordanoutlethomes.us, airjordanoutletinc.us, airjordanoutletmall.
us, airjordanoutletonline.us, airjordanoutletshop.us, airjordanoutletsite.us,
airjordanoutletstore.us

bhaappy0faiili.ru, bhaappy1loadzzz.ru, bhappy0sagruz.ru, bhappy1fajli.ru, bhappy2loaadz.ru,
bhappy3zagruz.ru, bhapy1file.ru, bhapy2fffilie.ru, bhapy3fajli.ru

fae412wdjjklpp.com, fae42wsdf.com, fae45223wed23.com, fae4523edf.com, fae452we334fvbmaa.com,
fae4dew2vb.com, faea2223dddfvb.com, faea22wsb.com, faea2wsxv.com, faeaswwdf.com

mbtshoes32.com, mbtshoesbetter.com, mbtshoesclear.com, mbtshoesclearancehq.com, mbtshoesdepot.
```

co.uk, mbtshoesfinder.com, mbtshoeslive.com, mbtshoesmallhq.com, mbtshoeson-deal.com, mbtshoesondeal.co.uk

tomshoesonlinestore.com, tomshoesoutletonline.net, tomshoesoutletus.com, tomsoutletsalezt.com, tomsoutletw.com, tomsoutletzt.com, tomsshoeoutletzt.com, tomsshoesonline4.com, tomsshoesonsale4.com, tomsshoesonsale7.com, tomsshoesoutlet2u.com

yahaoo.co.uk, yahho.jino.ru, yaho.co.uk, yaho.com, yahobi.com, yahoo.co.au, yahoo.cu.uk, yahoo.us, yahooi.aol, yahoon.com, yahooo.com, yahooo.com.mx, yahooz.com

(По-видимому, продажа подделок «фирменной» обуви является любимимым занятием спамеров.)

Корректные домены также могут быть представлены кластерами, обычно в интернациональных вариантах:

gigabyte.com, gigabyte.com.au], gigabyte.com.cn, gigabyte.com.mx, gigabyte.com.tr, gigabyte.com.tw, gigabyte.de, gigabyte.eu, gigabyte.fr, gigabyte.in, gigabyte.jp

hollywoodhairstyle.org, hollywoodhalfmarathon.com, hollywoodhereiam.com, hollywoodhiccups.com, hollywoodhomestead.com, hollywoodid.com, hollywoodilluminati.com, hollywoodlife.com, hollywoodmegastore.com, hollywoodmoviehd.com, hollywoodnews.com

pokerstars.com, pokerstars.cz, pokerstars.dk, pokerstars.es, pokerstars.eu, pokerstars.fr, pokerstars.gr, pokerstars.it, pokerstars.net, pokerstars.pl, pokerstars.pt

Поскольку многие домены, и корректные, и некорректные, не наблюдаются в кластерах, целью наших экспериментов будет достижение максимальной полноты некорректных доменов при сохранении высокой точности. Установим конкретные значения следующих параметров:

- кластер должен иметь размер не менее 10 элементов, чтобы считаться значимым;
- в кластере должно содержаться не менее 75 % доменов спама, чтобы пометить его как некорректный (мошеннический).

Выбор таких значений сведет к минимуму вероятность того, что корректные домены будут включены в кластеры, состоящие в основном из некорректных доменов.

Генерация кластеров

Приступим к первому этапу: разделение набора учетных записей или действий на группы элементов, похожих друг на друга. Мы будем рассматривать несколько различных методик и применим их к набору данных, содержащему домены спама.

Для кластеризации необходимо сгенерировать признаки используемых доменов. Признаки могут быть категориальными, числовыми или текстовыми (например, мешок слов). Для нашего примера выбраны следующие признаки:

- домен самого верхнего уровня (например, .com);
- процентное содержание букв, цифр и гласных в имени домена;
- возраст домена в днях в соответствии с датой регистрации сервисом whois;
- мешок слов, состоящий из n-грамм символов в имени домена (например, имя домена foo.com разбивается на 4-граммы [foo., oo.c, o.co, .com]) при значении n от 3 до 8;

Эту методику часто называют алгоритмом шинглов, или просто шинглингом (shingling). В следующем фрагменте кода на языке Python вычисляются n-граммы для отдельной строки:

```
def ngram_split(text, n):
  ngrams = [text] if len(text) < n else []
  for i in range(len(text)-n+1):
       ngrams.append(text[i:i+n])
  return(ngrams)
```

- первые n символов в имени домена, где n – одно из значений в последовательности (3, 5, 8)[1].

Правильный метод кластеризации сгенерирует относительно чистые кластеры (т. е. со значительным преобладанием корректных или некорректных объектов, а не состоящие из приблизительно одинакового количества тех и других). Кроме того, если данные асимметричны в значительной степени, как в нашем примере, алгоритм кластеризации должен восстановить баланс классов, насколько это возможно. Основная интуитивная предпосылка кластеризации состоит в том, что некорректные объекты встречаются в виде непропорционально плотных групп, поэтому если применять алгоритм кластеризации в условиях большей пропорциональности корректных кластеров по сравнению с некорректными, то от кластеризации, вероятнее всего, будет немного пользы.

Учитывая эти принципы при поиске наилучшей стратегии кластеризации, необходимо принять во внимание пропорциональную долю кластеров, помеченных как некорректные, пропорциональное соотношение доменов, помеченных как некорректные, внутри некорректных кластеров, а также характеристику полноты для некорректных кластеров.

Группирование

Метод группирования эффективнее всего применяется к признакам, которые могут принимать много различных значений, но не являются единственными в своем роде для каждого отдельного объекта. В нашем примере рассматриваются признаки n-грамм для применения метода группирования. Для каждого значения n от 3 до 8 домены группируются по каждой допустимой n-грамме. В табл. 6.2 показаны полученные результаты. (Напомним, что мы приняли минимальный размер кластера равным 10 объектам и 75 % спама в качестве порогового значения для пометки кластера как некорректного.)

Таблица 6.2. Результаты группирования в наборе данных, содержащих спам-домены, по n-граммам для различных значений n

n	Некорректные кластеры	Корректные кластеры	% некорректных кластеров	ИП домены	ЛП домены	Точность	Полнота
3	18	16 457	0.11%	456	122	0.79	0.03
4	95	59 954	0.16%	1518	256	0.86	0.11
5	256	72 343	0.35%	2240	648	0.78	0.16
6	323	52 752	0.61%	2176	421	0.84	0.16
7	322	39 390	0.81%	1894	291	0.87	0.14
8	274	28 557	0.95%	1524	178	0.90	0.11

[1] Это категориальный признак, который, в отличие от признака n-грамм, генерирует списки различной длины.

Здесь в столбцах «ИП домены» и «ЛП домены» приведено количество неповторяющихся спам-доменов и не спам-доменов соответственно внутри некорректных кластеров[1].

Из этих результатов понятно, что кластеризация по n-граммам в нашем конкретном случае не принесла особой пользы, так как представление некорректных кластеров занижено (они недостаточно полно представлены) по отношению к общей совокупности некорректных доменов (напомним, что некорректные домены составляют 2,7 % в общем наборе данных). Тем не менее относительно высокое значение полноты (особенно для n = 5, 6, 7) наводит на мысль о том, что необходимы дополнительные исследования, чтобы решить, можно ли даже при таких условиях создать классификатор для определения некорректных кластеров. Для дальнейших экспериментов и вычислений принимается значение n = 7.

Отметим также, что наш выбор признака в данном примере для группирования может привести к появлению доменов в нескольких кластерах. В этом случае при вычислении статистических характеристик очень важно исключить дублирующиеся домены, иначе можно получить завышенные значения точности и полноты. Если группирование выполняется по ключу, единственному в своем роде для каждого элемента, например по IP-адресу, с которого выполняется вход (регистрация), то проблемы дублирования доменов не возникает.

Чувствительное к местоположению хеширование (LSH)

Несмотря на то что группировка по одной n-грамме обеспечивает некоторую степень сходства между различными элементами кластера, весьма желательно применить более надежную концепцию сходства между элементами. Такой результат можно получить с помощью чувствительного к местоположению хеширования (locality-sensitive hashing – LSH). Из содержимого главы 2 вспомним, что чувствительное к местоположению хеширование (далее – LSH) аппроксимирует критерий (коэффициент) сходства Жаккара между двумя множествами. Если предположить, что такими множествами являются наборы n-грамм по именам доменов, то по методике Жаккара вычисляется процентная доля n-грамм, которые одинаковы в именах доменов, т. е. домены с совпадающими подстроками будут иметь высокий коэффициент сходства. Мы можем формировать кластеры, группируя домены с коэффициентом сходства, превышающим определенное пороговое значение.

Основным параметром для точной настройки метода LSH является пороговое значение сходства, используемое для формирования кластеров. Здесь имеет место классический компромисс между точностью и полнотой: слишком большие пороговые значения приводят к кластеризации только тех доменов, которые имеют высокие коэффициенты сходства, тогда как при малых пороговых значениях в кластеры включается больше элементов с низкими коэффициентами сходства, а количество кластеров увеличивается.

---

[1] Может возникнуть вопрос: каким образом сумма ИП (истинно положительных) доменов и ЛП (ложноположительных) доменов может быть меньше умноженного на 10 количества некорректных кластеров, если минимальный размер кластера равен 10? Дело в том, что некоторые домены могут присутствовать в нескольких кластерах, а при вычислении ИП/ЛП статистических характеристик дублирующиеся домены исключаются.

Кластеры определялись с использованием алгоритма minHash (*см.* главу 2) для списков n-грамм. Процедура кластеризации требует вычисления дайджеста (т. е. хеш-функции) для каждого набора n-грамм, затем для каждого домена dom проводится поиск всех доменов, дайджесты (значения хеш-функций) которых совпадают с дайджестом dom по требуемому количеству знаков (символов)[1]:

```
import lsh

def compute_hashes(domains, n, num_perms=32, max_items=100,
hash_function=lsh.md5hash):
    # domains - словарь объектов доменов, ключом является имя домена

    # Создание индекса LSH
    hashes = lsh.lsh(num_perms, hash_function)

    # Вычисление значений minHash
    for dom in domains:
        dg = hashes.digest(domains[dom].ngrams[n])
        domains[dom].digest = dg
        hashes.insert(dom, dg)
    return(hashes)

def compute_lsh_clusters(domains, hashes, min_size=10, threshold=0.5):
    # domains - словарь объектов доменов, ключом является имя домена
    # hashes - объект lsh, созданный процедурой compute_hashes

    clusters = []
    for dom in domains:
        # Собрать все домены, совпадающие с заданным дайджестом
        # result - словарь пар {domain : score}
        result = hashes.query(domains[dom].digest).
        result_domains = {domains[d] : result[d] for d in result
                if result[d] >= threshold}
        if len(result_domains) >= min_size:
            # Создать кластер с собранными в результате данными
            clusters.append(cluster(dom, result_domains))
    return(clusters)
hashes = compute_hashes(data, n, 32, 100)
clusters = compute_lsh_clusters(data, hashes, 10, threshold)
```

Для экономного использования памяти можно настроить структуру данных hashes так, чтобы ограничить количество элементов, сохраняемых по заданному дайджесту.

Эта реализация алгоритма использовалась для n с диапазоном значений от 3 до 7 с пороговыми значениями сходства (0,3, 0,5, 0,7). В табл. 6.3 приведены результаты работы алгоритма.

---

[1] Этот модуль содержит реализацию алгоритма minHash, его можно найти в репозитории кода **https://github.com/oreilly-mlsec/book-resources/tree/master/chapter6** для данной книги.

Таблица 6.3. Результаты работы алгоритма кластеризации по методу LSH, примененного к набору данных, содержащему домены спама, для различных размеров n-грамм и пороговых значений сходства

n	t = 0.3			t = 0.5			t = 0.7		
	Некорр. кластеры	% некорректн.	Полнота	Некорр. кластеры	% некорректн.	Полнота	Некорр. кластеры	% некорректн.	Полнота
3	24	2.4%	0.002	0	0.0%	0.000	0	0.0%	0.000
4	106	1.5%	0.013	45	12.9%	0.004	0	0.0%	0.000
5	262	1.8%	0.036	48	4.4%	0.004	0	0.0%	0.000
6	210	0.9%	0.027	61	4.0%	0.006	10	10	0.002
7	242	1.0%	0.030	50	2.7%	0.004	38	38	0.003

По табл. 6.3 можно видеть, что при увеличении порогового значения сходства алгоритм обнаруживает меньше кластеров, но сформированные кластеры в среднем хуже по качеству.

Метод k-средних

Когда люди задумываются о кластеризации, первое, что большинству из них приходит в голову, – метод k-средних. Алгоритм k-средних обеспечивает эффективные вычисления и легок для понимания. Но обычно это не самый лучший алгоритм для выявления нарушений. Главная проблема состоит в том, что метод k-средних требует предварительного точного определения количества кластеров k. Поскольку заранее невозможно определить предполагаемое количество некорректных и легитимных кластеров, лучшее, что можно сделать, – установить значение k равным количеству точек данных, разделенному на ожидаемое количество точек данных в некорректном кластере, и надеяться, что алгоритм создаст кластеры правильного размера.

Вторая проблема – каждый элемент набора данных распределяется в какой-либо кластер. В результате если значение k слишком мало, то даже элементы, не очень похожие друг на друга, будут искусственно объединены в кластеры. И наоборот, если значение k слишком велико, в итоге будет создано множество мелких кластеров, следовательно, будет утеряно главное преимущество кластеризации. С другой стороны, если используется группирование или хеширование, то многие элементы просто не будут объединены с какими-либо другими элементами, и вам придется сосредоточить свои усилия на действительно существующих кластерах.

Третья проблема, ранее упоминаемая в главе 2, заключается в том, что метод k-средних не работает с категориальными признаками и только иногда успешно работает с бинарными признаками. Поэтому если в наборе данных много бинарных и/или категориальных признаков, то эффективность различения данных с помощью этого алгоритма резко снизится.

Чтобы продемонстрировать все перечисленные проблемы на практике, мы обработали алгоритмом k-средних наш набор данных с доменами спама с различными значениями k. Поскольку категориальные признаки необходимо было исключить из работы, мы оставили только значения процентного содержания букв, чисел и цифр, а также дату регистрации домена по данным whois. В табл. 6.4, как и ожидалось, мы видим лишь несколько некорректных кластеров, сформированных в результате работы метода k-средних.

Таблица 6.4. Результаты кластеризации с применением метода k-средних к набору данных, содержащему домены спама

k = кол-во кластеров	Некорректные кластеры	ИП домены	ЛП домены	Точность	Полнота
100	0	0	0	–	0
500	0	0	0	–	0
1000	1	155	40	0.79	0.011
5000	4	125	28	0.82	0.009
10 000	10	275	58	0.83	0.020

Более того, здесь можно заметить, что с увеличением значения k не наблюдается рост эффективности различения корректных и некорректных кластеров, – доля некорректных кластеров постоянно колеблется около 0,1 % для всех использованных в примере значений k.

Оценка кластеров

В предыдущем разделе рассматривалось практическое применение нескольких методик для поиска и формирования кластеров похожих доменов в нашем наборе данных. Но сама по себе операция кластеризации не позволяет сразу достичь установленной цели – выявления нарушений, – кластеризация просто реорганизует данные в форме, которая облегчает обнаружение объектов-нарушителей. Следующий этап – исследование кластеров и определение, является каждый кластер корректным или некорректным.

Вообще говоря, при кластеризации некорректных объектов сразу может возникнуть мысль, что каждый кластер определенного размера автоматически является некорректным. Подобные правила представляют собой неплохой отправной пункт, но если ваш сайт весьма популярен и содержит большой объем данных, то неизбежны промахи, например:

- слишком много операций выполняется с одного IP-адреса? Этот адрес может быть мобильным шлюзом;
- множество учетных записей совместно использует отслеживающие закладки cookie? Возможно, все эти учетные записи созданы на одном компьютере общего пользования;
- слишком быстрые последовательности процедур регистрации с адресов электронной почты в одном и том же формате? Это может быть урок в школе или подготовительные курсы в организации, когда всем учащимся предлагают задание по созданию учетной записи.

Как отличить корректные кластеры от некорректных? Ответ все тот же: по данным. Необходимо извлечь признаки уровня кластера, которые позволяют различать два типа кластеров. Здесь предварительная, интуитивная идея состоит в том, что если ответственность за создание кластера лежит на единственном действующем лице, то данные в этом кластере, вероятнее всего, имеют необычное распределение по некоторому измерению. Простой пример: если обнаружена группа учетных записей с одним и тем же именем, то такая группа вызывает подозрение, если распределение имен в кластере приблизительно соответствует распределению имен по всему сайту, то такая группа менее подозрительна (по крайней мере, в отношении распределения имен).

В идеальном случае этап оценки кластера следовало бы считать задачей обучения с учителем. Это означает, что необходимо получить метки уровня кластера и вычислить (определить) признаки уровня кластера, которые можно передать как входные данные в стандартный алгоритм классификации, такой как логистическая регрессия или случайный лес. Рассмотрим этот процесс.

Присваивание меток

Если учетные записи сгруппированы в кластеры, но имеют только метки уровня учетных записей, необходимо разработать процедуру, которая объединяет метки уровня учетных записей в метки для кластеров. Простейший метод – определение большинства при голосовании: если большинство учетных записей в кластере некорректно, то и кластер объявляется некорректным. Для обобщения можно установить любое пороговое значение t для процедуры присваивания метки и помечать кластер как некорректный, если процент некорректных меток в кластере превышает значение t. В нашем примере выявления доменов спама выбрано значение t = 0,75.

В некоторых ситуациях, когда необходимы еще более жесткие условия, кластер помечается как некорректный сразу же после обнаружения в нем единственного некорректного элемента. Например, если группируются рекламные домены на основе их страниц размещения, то при обнаружении единственного мошеннического объекта, указывающего на конкретную страницу размещения, вероятнее всего, необходимо пометить как мошеннические все рекламные объекты, ссылающиеся на эту страницу.

Извлечение признаков

После присваивания меток следует объединить признаки уровня учетной записи в признаки уровня кластера, для того чтобы представить каждый кластер в виде одного числового вектора, который можно передать в классификатор. Пользуясь нашим интуитивным предположением о том, что некорректные кластеры будут показывать меньшее различие по определенным измерениям, необходимо вычислить признаки для числового измерения такого различия. Среди числовых признаков уровня учетной записи мы выбрали девять признаков уровня кластера для вычислений:

- минимальное, максимальное, срединное значения, квартили;
- среднее значение и стандартное отклонение;
- процент нулевых (null или zero) значений.

Для категориальных признаков уровня учетных записей вычисляются четыре признака:

- количество различающихся значений;
- процент значений, принадлежащих моде;
- процент нулевых значений;
- энтропия.

Рассмотрим несколько конкретных примеров реализации этого процесса с использованием кластера n-грамм, соответствующих экземплярам корректных и некорректных кластеров, которые были определены ранее. По полученным ранее результатам анализа следует сосредоточить внимание на 7-граммах. На рис. 6.2 показаны признаки по каждому домену для имен доменов, содержащих

7-грамму «jordano». На рис. 6.3 отображены аналогичные признаки для доменов, содержащих 7-грамму «gabyte».

	domain	first3	first5	first8	label	length	pct_digits	pct_letters	pct_vowels	tld	whois
0	airjordanoutletonline.us	air	airjo	airjorda	1	24	0	0.958333	0.458333	us	17036
1	airjordanoutletgroup.us	air	airjo	airjorda	1	23	0	0.956522	0.434783	us	None
2	airjordanoutletcenter.us	air	airjo	airjorda	1	24	0	0.958333	0.416667	us	None
3	airjordanoutletwork.us	air	airjo	airjorda	1	22	0	0.954545	0.409091	us	None
4	airjordanoutletmall.us	air	airjo	airjorda	1	22	0	0.954545	0.409091	us	None
5	airjordanoutletusa.us	air	airjo	airjorda	1	21	0	0.952381	0.47619	us	None
6	airjordanoutletinc.us	air	airjo	airjorda	1	21	0	0.952381	0.428571	us	None
7	airjordanoutletclub.us	air	airjo	airjorda	1	22	0	0.954545	0.409091	us	None
8	autoairjordanoutlet.us	aut	autoa	autoairj	1	22	0	0.954545	0.5	us	None
9	airjordanoutlethomes.us	air	airjo	airjorda	1	23	0	0.956522	0.434783	us	None
10	airjordanochaussure.com	air	airjo	airjorda	1	23	0	0.956522	0.434783	com	None
11	airjordanoutletdesign.us	air	airjo	airjorda	1	24	0	0.958333	0.416667	us	None
12	belleairjordanoutlet.us	bel	belle	belleair	1	23	0	0.956522	0.434783	us	None
13	airjordanoutletstore.us	air	airjo	airjorda	1	23	0	0.956522	0.434783	us	17036
14	airjordanoutletshop.us	air	airjo	airjorda	1	22	0	0.954545	0.409091	us	None
15	allairjordanoutlet.us	all	allai	allairjo	1	21	0	0.952381	0.428571	us	None
16	airjordanoutletsite.us	air	airjo	airjorda	1	22	0	0.954545	0.454545	us	None

Рис. 6.2 ❖ Имена доменов, содержащие 7-грамму «jordano»

	domain	first3	first5	first8	label	length	pct_digits	pct_letters	pct_vowels	tld	whois
0	gigabyte.fr	gig	gigab	gigabyte	0	11	0	0.909091	0.272727	fr	11023
1	gigabyte.cn	gig	gigab	gigabyte	0	11	0	0.909091	0.272727	cn	12128
2	gigabyte.jp	gig	gigab	gigabyte	0	11	0	0.909091	0.272727	jp	11407
3	gigabyte.de	gig	gigab	gigabyte	0	11	0	0.909091	0.363636	de	14350
4	gigabyte.com.cn	gig	gigab	gigabyte	0	15	0	0.866667	0.266667	cn	10787
5	gigabyte.com.tr	gig	gigab	gigabyte	0	15	0	0.866667	0.266667	tr	None
6	gigabyte.pt	gig	gigab	gigabyte	0	11	0	0.909091	0.272727	pt	12982
7	gigabyte.asia	gig	gigab	gigabyte	0	13	0	0.923077	0.461538	asia	13886
8	gigabyte.in	gig	gigab	gigabyte	0	11	0	0.909091	0.363636	in	None
9	gigabyte.ru	gig	gigab	gigabyte	0	11	0	0.909091	0.363636	ru	10928
10	gigabyte.com.au	gig	gigab	gigabyte	0	15	0	0.866667	0.4	au	None
11	gigabyte.com.tw	gig	gigab	gigabyte	0	15	0	0.866667	0.266667	tw	9982
12	gigabyte.tw	gig	gigab	gigabyte	0	11	0	0.909091	0.272727	tw	13083
13	gigabyte.com.mx	gig	gigab	gigabyte	0	15	0	0.866667	0.266667	mx	12750
14	gigabyte.eu	gig	gigab	gigabyte	0	11	0	0.909091	0.454545	eu	None
15	gigabyte.co.za	gig	gigab	gigabyte	0	14	0	0.857143	0.357143	za	None
16	gigabyte.pl	gig	gigab	gigabyte	0	11	0	0.909091	0.272727	pl	None
17	gigabyte.com	gig	gigab	gigabyte	0	12	0	0.916667	0.333333	com	9903

Рис. 6.3 ❖ Имена доменов, содержащие 7-грамму «gabyte»

Развертывая 5 числовых и 4 категориальных признака в соответствии с приведенными выше указаниями, получаем всего 65 признаков. Например, признак уровня домена «whois» (позволяющий получить возраст домена в днях) генерирует признаки уровня кластера, показанные на рис. 6.4.

	whois_max	whois_mean	whois_median	whois_min	whois_pct_null	whois_pct_zero	whois_q1	whois_q3	whois_std
jordano	17036.0	17036.000000	17036.0	17036.0	0.882353	0.0	17036.00	17036.00	0.00000
gabyte.	14350.0	11934.083333	11767.5	9903.0	0.333333	0.0	10892.75	13007.25	1481.39728

Рис. 6.4 ❖ Примеры признаков уровня кластера для «whois»

Признак «top-level domain» генерирует признаки уровня кластера, показанные на рис. 6.5.

	tld_entropy	tld_num_unique	tld_pct_mode	tld_pct_null	tld_pct_unique
jordano	0.322757	2.0	0.941176	0.0	0.117647
gabyte.	3.947703	16.0	0.111111	0.0	0.888889

Рис. 6.5 ❖ Примеры признаков уровня кластера для «top-level domain»

Из приведенных выше двух примеров можно понять, что whois главным образом возвращает нулевой (null) результат для некорректных доменов, в то время как для корректных доменов возвращаются результаты с широким диапазоном числовых значений. Это признак, вполне пригодный для определения различия. Кроме того, предполагается, что большое разнообразие имен доменов верхнего уровня характерно для корректного кластера.

Классификация

Теперь попробуем преобразовать все приведенные выше интуитивные соображения и предположения в практическую реализацию для тренировки классификатора. Воспользуемся классификатором на основе случайного леса, являющегося нелинейным классификатором, демонстрирующим эффективность по умолчанию с минимальной тонкой настройкой. Здесь выполнена субдискретизация корректных кластеров в тренировочном наборе, чтобы не перегружать классификатор и не создавать предвзятости, но тестовый набор не изменен, чтобы обеспечить достоверные вычисления точности и полноты:

```
from sklearn.metrics import roc_auc_score, roc_curve, precision_recall_curve
from sklearn.model_selection import train_test_split
from sklearn.ensemble import RandomForestClassifier
from random import random
import matplotlib.pyplot as plt

# Добавление случайного объекта в каждую строку, с использованием для выборки
R = [random() for i in range(len(ngram_cluster_features))]
ngram_cluster_features['rand'] = R

# Разделение данных: 2/3 тренировочный набор, 1/3 тестовый набор
# и субдискретизация корректных кластеров
train, test = train_test_split(ngram_cluster_features.fillna(value=0),
                               test_size=0.33)
sample_factor = 0.2
```

```
sampled_train = train[(train.label == 1) | (train.label == 0) &
                      (train.rand < sample_factor)]

# Передача данных и прогнозирование
features = sampled_train[sampled_train.columns.difference(
    ['label','rand','score'])]
labels = sampled_train.label
clf = RandomForestClassifier(n_estimators=20)
clf.fit(features, labels)
probs = clf.predict_proba(test[train.columns.difference(
    ['label','rand','score'])])

# Вычисление и построение кривой точность-полнота (P-R)
precision, recall, thresholds = precision_recall_curve(
    test.label, probs[:,1])
plt.step(recall, precision, color='b', alpha=0.2, where='post')
plt.fill_between(recall, precision, step='post', alpha=0.2, color='b')
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.ylim([0.0, 1.05])
plt.xlim([0.0, 1.0])
plt.title('Precision-Recall curve for 7-gram groupings')
plt.show()

# Поиск порогового значения для 95 % точности
m = min([i for i in range(len(precision)) if precision[i] > 0.95])
p,r,t = precision[m], recall[m], thresholds[m]
print(p,r,t)
```

По результатам этих вычислений и по кривой «точность–полнота» для данного классификатора, показанной на рис. 6.6, можно понять, что можно достичь 61 % полноты и 95 % точности в кластерах с пороговым значением 0,75 (т. е. 15 из 20 деревьев в используемом лесу классифицируют кластер как некорректный).

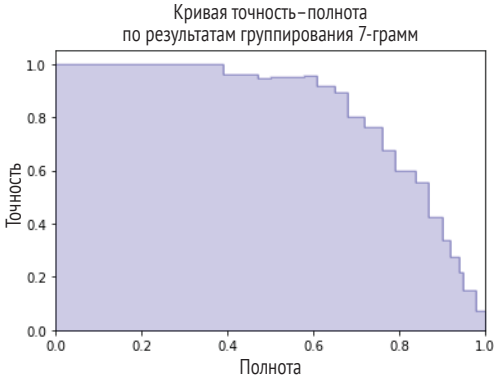


Рис. 6.6 ❖ Кривая «точность–полнота» для группирования по 7-граммам при классификации доменов спама

Но это вычисление выполнено на уровне кластера, а как обстоят дела с точностью и полнотой на уровне отдельного домена? Например, если ложноположительные результаты наблюдаются в большем количестве в среднем по класте-

рам, чем истинно положительные результаты, то точность по отдельным доменам ниже, чем на уровне кластеров.

Возможно также вычисление точности и полноты на уровне элементов, как показано ниже:

```
# Вычисление точности/полноты на уровне элементов
pos = (test.score * test['count'])
neg = (1-test.score) * (test['count'])
tp = sum(pos[test.label >= t])
fp = sum(neg[test.label >= t])
tn = sum(neg[test.label < t])
fn = sum(pos[test.label < t])
item_precision = 1.0*tp/(tp+fp)
item_recall = 1.0*tp/(tp+fn)
```

В этом примере обнаруживается, что точность немного снизилась до 92 %, а полнота буквально «упала» до 21 %. Такой результат можно было предположить заранее с учетом того, что многие некорректные домены в исследуемом наборе данных не являются частью кластеров, следовательно, должны быть выявлены с помощью каких-либо других средств.

Дальнейшие направления кластеризации

В примере из предыдущего раздела было показано, как используются различные алгоритмы для генерации кластеров по набору данных, выбранному для этого примера, а также как программно определить по этим данным, какие кластеры являются некорректными. При реализации вашей собственной системы кластеризации доступно несколько направлений, по которым можно усовершенствовать этот пример:

- эксперименты с различными методами кластеризации, в частности с описанными в главе 2;
- эксперименты с различными классификаторами и параметрами классификаторов;
- добавление новых признаков на уровне элементов;
- добавление новых объединенных признаков на уровне кластера;
- выборка данных, для того чтобы «полукорректные» кластеры (т. е. кластеры, в которых пропорциональная доля некорректных элементов достаточно близка к пороговому значению, что позволяет объявить такой кластер некорректным) оказывали меньшее воздействие на итоговый результат;
- присваивание дополнительного веса элементам, присутствующим в нескольких корректных или некорректных кластерах;
- добавление второго классификатора для выявления ложноположительных элементов в кластере (например, если 19 из 20 элементов в кластере являются некорректными, а 20-й явно корректен, то можно ли автоматически определить этот корректный элемент?).

Как и для многих других аспектов обеспечения безопасности, наилучшие идеи для дальнейшей работы появляются в процессе исследования данных. Какие тренды (тенденции) не были учтены? Какие признаки можно использовать для идентификации ошибок классификатора? В чем ложноположительные и ложно-

отрицательные результаты похожи друг на друга? Если вы точно и профессионально ответите на эти вопросы, то сможете воспользоваться практическим материалом данной главы для усовершенствования алгоритмического решения конкретной задачи.

Резюме

Потребительская веб-среда (и связанные с ней приложения) предоставляет огромный диапазон поверхностей атак, которые злоумышленники могут использовать для монетизации. В большинстве случаев атака потребует одной или нескольких учетных записей, поэтому защита от получения атакующими доступа к учетным записям может остановить многие различные типы атак. В этой главе рассматривались способы защиты от захвата учетных записей и от создания мошеннических учетных записей. Это два основных способа, которыми злоумышленники могут получить доступ к необходимым для них учетным записям. Также рассматривались способы выявления финансовых мошенничеств и их автоматизации – двух широко распространенных методик, используемых для атак на практически любой реально эксплуатируемый объект.

Применение машинного обучения для задач выявления нарушений связано с собственным, присущим только ему набором трудностей: собрать по-настоящему достоверные практические «полевые» данные нелегко, а модели непременно должны поддерживать хрупкий баланс между исследованием того, что уже известно, и обнаружением новых методик атак. Мы подробно рассмотрели некоторые из крупных затруднений в процессе применения машинного обучения для выявления нарушений и предложили некоторые варианты решений по их устранению или обходу.

Несмотря на то что методы обучения с учителем являются чрезвычайно мощным средством решения задач по выявлению нарушений, во многих случаях можно использовать методы кластеризации для нейтрализации атак любого масштаба. В сочетании с методами обучения с учителем эти методы могут позволить определить и блокировать крупномасштабные атаки весьма быстро. Некоторые методы кластеризации были продемонстрированы на практических работающих примерах классификации имен доменов, являющихся источниками спама.

До настоящего момента излагаемый материал был в определенной степени теоретическим: сбор определенных сигналов, реализация определенных алгоритмов и выявление злоумышленников и их действий. Но в реальной практике все не так просто и ясно. В главе 7 основное внимание уделяется проблемам, возникающим при внедрении идей, рассмотренных в предыдущих главах, в реально эксплуатируемые системы.

Глава 7

Производственные системы

До настоящего момента в этой книге основное внимание уделялось реализации алгоритмов машинного обучения для обеспечения безопасности в изолированных лабораторных условиях. После того как вы убедились в том, что выбранный алгоритм работает, следующим этапом, вероятнее всего, становится выбор программного обеспечения (ПО), готового для промышленной эксплуатации. Развертывание систем машинного обучения в реальной эксплуатационной среде связано с разнообразными затруднениями, с которыми вы, возможно, не сталкивались на этапах экспериментирования и разработки. Что означает для инженера по эксплуатации действительно масштабируемая система машинного обучения? Как управлять работоспособностью, надежностью и актуальностью служб обеспечения безопасности веб-уровня в динамических средах с постоянными изменениями? В этой главе рассматривается обеспечение безопасности и обработка данных в условиях реальной производственной эксплуатации, и мы попытаемся ответить на эти и некоторые другие вопросы.

Начнем с точного определения того, что означает для систем машинного обучения полная готовность к производственной эксплуатации, способность к быстрому развертыванию и масштабируемость.

Определение зрелости и масштабируемости систем машинного обучения

Вместо перебора абстрактных терминов для описания качества кода, предназначенного для производственной эксплуатации, гораздо полезнее подробно рассмотреть некоторые характеристики, которыми должны обладать зрелые и масштабируемые системы машинного обучения. Приведенный ниже список признаков и характеристик описывает идеальную систему машинного обучения вне зависимости от того, предназначена ли она для обеспечения безопасности. Пункты, выделенные полужирным шрифтом, особенно важны для систем машинного обучения, специально предназначенных для обеспечения безопасности. Кроме того, список представляет собой план изложения материала в следующих разделах текущей главы, поэтому если вы заинтересовались каким-либо пунктом списка, то можете сразу перейти к чтению соответствующего раздела. Рассматриваются следующие темы:

- качество данных:
  - **непредвзятые данные;**

  - **данные из реальной практики, которые можно проверить;**
  - эффективное восполнение недостающих данных;
- качество модели:
  - **эффективная оптимизация гиперпараметров;**
  - **A/B-тестирование моделей;**
  - регулярные циклы обратной связи;
  - воспроизводимые результаты;
  - объяснимые результаты;
- эффективность:
  - тренировка и прогнозирование с минимальными задержками;
  - **масштабируемость (т. е. способность обрабатывать возросший в 10 раз трафик);**
  - **автоматизированный и эффективный сбор данных;**
- удобство сопровождения:
  - создание контрольных точек проверки и версий моделей;
  - **отлаженный процесс развертывания модели;**
  - устойчивость к ошибкам (амортизация отказов);
  - **простота конфигурирования и точной настройки;**
  - **подробная документация;**
- мониторинг и оповещение:
  - **мониторинг работоспособности и рабочего процесса системы;**
  - мониторинг эффективности системы (т. е. точности/полноты);
  - мониторинг распределения данных (например, изменения поведения пользователей или адаптация противника к изменениям в системе);
- безопасность и надежность:
  - устойчивость и надежность работы во враждебных средах;
  - **обеспечение надежной защиты секретных данных.**

Это длинный список, но не все перечисленные в нем пункты применимы для всех типов систем. Например, объяснимые результаты не обязательны для системы рекомендаций просмотра онлайнового видео, так как обычно такая система не дает каких-либо существенных гарантий, а цена неверного прогноза весьма мала. Системы, которые по своей сущности обычно не привлекают мошенников, могут не уделять особого внимания выделению ресурсов на укрепление сопротивляемости модели враждебной деятельности.

Важные аспекты систем машинного обучения для обеспечения безопасности

Приложения машинного обучения для обеспечения безопасности непременно должны соответствовать ряду обязательных требований, выполняемых до ввода таких систем в реальную эксплуатацию. С самого начала почти от всех подобных систем требуется высокая точность прогнозов, поскольку цена ошибок в этой области деятельности слишком высока. Коэффицент ошибок 0,001 (т. е. точность прогнозов 99,9 %) может быть достаточным для модели планирования продаж, которая выполняет 100 прогнозов в день, – в среднем только 1 неверный прогноз будет появляться за каждые 10 дней. С другой стороны, классификатор сетевых пакетов, проверяющий миллион TCP-пакетов в секунду, будет неправильно

классифиицировать 1000 пакетов каждую минуту. Без отдельно организованных процессов фильтрации таких ложноположительных и дожноотрицательных результатов коэффициент ошибок 0,001 не приемлем для этого типа систем. Если каждый ложноположительный результат должен вручную обрабатываться и классифицироваться человеком-аналитиком, то стоимость операции становится слишком высокой. Для каждого ложноотрицательного результата или невыполненного выявления последствия могут стать роковыми: вся система будет скомпрометирована.

Все перечисленные выше свойства зрелости и масштабируемости систем машинного обучения важны, но выделенные пункты являются наиважнейшими для достижения успеха в применении систем машинного обучения для обеспечения безопасности.

Рассмотрим подробнее этот список характеристик, а также специализированные методики разработки масштабируемых, эффективных и качественных систем машинного обучения для обеспечения безопасности. В следующих разделах рассматриваются наиболее часто возникающие проблемы или наиболее трудные задачи практического применения методов машинного обучения для обеспечения безопасности. В каждом конкретном случае сначала описывается задача или цель и объясняется, почему она важна. Потом рассматриваются способы и методики проектирования системы, которая может помочь в достижении поставленной цели или смягчить проблему.

Качество данных

Качество входных данных для систем машинного обучения определяет успех или провал. При тренировке классификатора спама в электронной почте с использованием методов обучения с учителем передача в алгоритм тренировочных данных, которые содержат только спам, рекламирующий медицинские и оздоровительные средства, не позволит создать сбалансированную обобщенную модель. Полученная в итоге система сможет успешно распознавать невостребованные сообщения, рекламирующие средства избавления от лишнего веса, но, вероятнее всего, окажется неспособной определять спам других типов.

Проблема: необъективность данных

Хорошо сбалансированные наборы данных встречаются редко, а использование несбалансированных наборов данных может привести к необъективности результатов, которую трудно выявить. Например, наборы данных о вредоносном ПО редко бывают достаточно разнообразными, для того чтобы охватить все типы вредоносного ПО, которые предполагается классифицировать с помощью создаваемой конкретной системы. В зависимости от того, что было собрано в ловушках, от выборок из собранных данных, от источников невредоносных бинарных файлов и т. д. в таких наборах данных часто существует значительная необъективность.

Алгоритмам машинного обучения необходимы предоставляемые им наборы данных для выполнения задачи обучения. Термин «совокупность» (population) используется для обозначения всего объема данных, характеристики и/или поведение которых должен смоделировать алгоритм. Например, предположим, что

необходимо применить алгоритм машинного обучения для отделения всех фишинговых сообщений электронной почты от обычных корректных сообщений. В этом случае совокупность означает, что все сообщения электронной почты, которые существовали в прошлом, существуют в настоящем и будут существовать в будущем.

Поскольку обычно невозможно собрать экземпляры из всей полной совокупности, наборы данных создаются с помощью выборок данных из источников, предоставляющих экземпляры, принадлежащие к интересующей нас совокупности. Например, предположим, что подходящий набор данных для организации X – это сообщения электронной почты с корпоративного e-mail-сервера за март. Использование такого набора данных в итоге позволит создать классификатор, хорошо работающий для организации X, но нет никаких гарантий, что он будет демонстрировать такую же высокую эффективность и в дальнейшем или если будет применен в другой организации. Фишинговые сообщения, получаемые другой организацией Y, принадлежат к той же совокупности, но их характеристики могут абсолютно отличаться от характеристик, выявленных в организации X, поэтому переданный классификатор вряд ли продемонстрирует хорошие результаты для набора сообщений электронной почты организации Y. Предположим также, что экземпляры фишинговых сообщений в наборе данных в основном представляют собой подделки налоговых требований и собственно фишинговые сообщения с учетом того, что март и апрель в США – это период уплаты налогов. Если не предпринять особых мер, то модель может не обучиться определению характеристик других типов фишинговых сообщений и, вероятнее всего, не покажет достаточной эффективности на обобщенных тестах. Поскольку была поставлена задача создания универсального классификатора фишинга, успешно работающего со всеми сообщениями электронной почты, набор данных, примененный для тренировки, был неправильно выбран из всей совокупности данных. Такой классификатор является жертвой необъективности выбора (selection bias) и необъективности исключения (exclusion bias)[1] из-за временны́х и контекстных эффектов, влияющих на выбор конкретного набора данных, используемого для тренировки классификатора.

Необъективность выбора и необъективность исключения – это распространенные формы необъективности, причиной которых могут являться дефекты в процессах (потоках) сбора данных. Эти формы необъективности возникают из-за систематического неправильного выбора или исключения данных из совокупности, предназначенной для анализа. В результате формируются наборы данных, обладающие свойствами и распределением, совершенно не характерными для исследуемой совокупности.

Необъективность исследователя (observer bias), или эффект ожидания исследователя (observer-expectancy effect), представляет собой другой часто встречающийся тип необъективности, возникающий вследствие ошибок человеческих

[1] Отметим, что необъективность (bias), или алгоритмическая необъективность (algorithmic bias), в статистике и машинном обучении также является термином, используемым для описания ошибок в предположениях, сделанных алгоритмом обучения, которые могут привести к недоподгонке (underfit) алгоритма. Здесь этот термин используется в другом смысле – им обозначается необъективность данных (data bias), т. е. несоответствие набора данных общему представлению совокупности.

предположений или процессов, проектируемых людьми. Процессы извлечения признаков из бинарных программных файлов могут быть необъективными по отношению к определенным типам поведения, демонстрируемым бинарными файлами, на поиск которых специально ориентирована тренировка людей-аналитиков. Например, DNS-запросы к серверам контроля и управления. В результате механизмы выявления и сбора в таких конвейерных процессах могут оставлять без внимания другие не менее важные, но не так часто проявляющиеся вредоносные действия, такие как случаи неавторизованного прямого доступа к памяти. Подобная необъективность приводит к неактуальности и неточности данных и к назначению неправильных меток для экземпляров, что отрицательно влияет на точность всей системы в целом.

Проблема: неточность меток

При обучении с учителем неправильно помеченные данные становятся причиной потери точности алгоритмов машинного обучения. Проблема усугубляется, если валидационные наборы данных также снабжены неправильными метками. Проверка (валидация) точности во время разработки может выглядеть многообещающе, но модель, вероятнее всего, не будет работать так, как ожидалось при обработке реальных данных в производственном эксплуатационном режиме. Проблема неточности меток чаще всего возникает при массовом беспорядочном сборе всех доступных данных без принятия правильных защитных мер. Обратная связь с пользователями или экспертами, принимающими решения на основе неполной информации, также может привести к некорректной разметке данных. Данные с некорректными метками могут значительно искажать цели обучения алгоритмов, если их вовремя не распознать и не скорректировать.

Проверка корректности меток в наборе данных часто требует больших ресурсов и трудозатрат людей-экспертов. Проверка может занимать несколько часов работы профессионала в области обеспечения безопасности, для того чтобы определить, действительно ли метки бинарных файлов, присвоенные системой машинного обучения, соответствуют их вредоносному поведению. Даже процедура валидации случайно выбранных подмножеств из основных наборов данных может стать весьма затратной.

Решения: качество данных

Существует множество причин возникновения проблем с качеством данных, но имеется также и несколько способов простого и быстрого устранения этих проблем. Наиболее важным этапом в устранени проблем с качеством данных в системах машинного обучения, предназначенных для обеспечения безопасности, является определение того, что проблема действительно существует. Дисбаланс классов (рассматриваемый в главе 5) представляет собой проявление необъективности данных, при котором количество экземпляров одного класса данных значительно меньше (или больше) количества экземпляров другого класса. Дисбаланс классов – это вполне очевидная проблема, которую можно обнаружить на этапе исследования или тренировки и устранить с помощью выборки с запасом или субдискретизации, как было описано ранее. Но существуют и другие формы необъективности и неточности данных, которые могут быть малозаметными, но причиняющими не меньший ущерб эффективности модели. Выявление необъективности выбора

и необъективности исследователя является трудной задачей, особенно если корни проблемы скрываются в недостаточной проработанности проектных решений и реализации. Затраты времени и ресурсов для точного понимания целей поставленной задачи и природы данных – это единственный способ узнать, существуют ли еще какие-либо важные аспекты данных, включенных в рабочие наборы, которые остались неохваченными (не определенными должным образом).

В некоторых случаях можно избежать проблем с качеством данных, тщательно определяя рабочую область («область видимости») задачи. Например, системы, предназначенные для выявления всех типов фишинговых сообщений электронной почты, будут иметь весьма ограниченное время для генерации репрезентативного тренировочного набора данных. Но если сократить рабочую область этой общей задачи до устранения самой главной проблемы, возникшей в конкретной организации, – например, выявление фишинговых сообщений, которые пытаются спровоцировать пользователя на кликджек (clickjack)[1], – намного проще собрать более точно определенные данные для этой задачи.

Неточности меток из-за ошибок в процессе присваивания меток человеком можно сделать менее вероятными, если привлечь к этому процессу нескольких независимых специалистов. Можно воспользоваться статистическими метриками (такими как каппа-коэффициент Флейсса (Fleiss' kappa)[2]) для оценки надежности согласования между несколькими экспертами, назначающими метки и исключения некорректных меток. Предполагая, что метки были назначены специалистами без ошибок и без злого умысла, уровень несогласованности между людьми-экспертами по конкретной метке экземпляра выборки также часто используется как верхняя граница правдоподобия, с которым классификатор на основе машинного обучения способен спрогнозировать корректную метку для этого экземпляра. Например, предположим, что два независимых эксперта помечают некоторое сообщение электронной почты как спам, а два других эксперта считают, что это «правильное» сообщение. Значит, класс данного сообщения нельзя определить однозначно, если даже люди-эксперты не могут прийти к единому мнению о его метке. Классификаторы на основе машинного обучения, вероятнее всего, не будут эффективно работать с такими экземплярами, поэтому наилучшее решение – исключить подобные экземпляры из набора данных, чтобы избежать путаницы в целях обучения для алгоритма.

Если известно, что в наборе данных есть шумовые (неточные) метки, но невозможно или слишком затратно устранить все неточности, то заслуживающим внимания компромиссом может стать увеличение параметров регуляризации для преднамеренного сдерживания переподгонки за счет точности прогнозов. Переподгонка

---

[1] Кликджекинг (clickjacking) – это метод веб-атаки, позволяющий обманным путем заставить пользователя выполнить клик по объекту, отличающемуся от объекта, который подразумевает пользователь. Обычно для этого пользователю предлагается ложный интерфейс взамен настоящего. Кликджекинг заставляет пользователей совершать непреднамеренные действия, выгодные для атакующего, например раскрывать секретную информацию, предоставлять доступ к некоторым защищенным ресурсам или инициировать вредоносные операции.

[2] *J. L. Fleiss and J. Cohen*. The Equivalence of Weighted Kappa and the Intraclass Correlation Coefficient as Measures of Reliability. Educational and Psychological Measurement 33 (1973): 613–619.

модели к набору данных с неточными метками может привести к катастрофическим результатам, определяемым схемой «мусор на входе, мусор на выходе».

Проблема: отсутствующие (потерянные) данные

Отсутствующие (потерянные) данные – одна из наиболее часто возникающих проблем в процессе машинного обучения. Весьма часто в наборах данных встречаются строки с отсутствующими (пропущенными) значениями. Причиной этого могут быть ошибки в процессе сбора данных, но отсутствие или потеря данных в наборах также может объясняться ошибками при проектировании. Например, если набор данных формируется по результатам опросов людей-респондентов, то в список могут быть включены необязательные вопросы, на которые некоторые люди предпочитают не отвечать. Из-за этого в набор данных включаются нулевые значения, создающие проблемы на этапе анализа. Некоторые алгоритмы отказываются классифицировать строки с нулевыми значениями, определяя их как не имеющие смысла (бесполезные), даже если такие строки содержат корректные данные в большинстве столбцов. Другие алгоритмы подставляют значения по умолчанию во входные или выходные данные, но такой подход может привести к ошибочным результатам.

Самая распространенная ошибка – заполнение пропусков в данных контрольными значениями (sentinel values), т. е. фиктивными данными того же формата, что и остальная часть столбца. Такое фиктивное значение сообщает оператору, что изначально это значение было пустым, например 0 или –1 для числовых значений. Контрольные значения загрязняют набор данных, вставляя данные, не представляющие исходную совокупность, из которой формировалась рабочая выборка экземпляров. Для человека может быть вполне понятно, что 0 или –1 не являются корректными допустимыми значениями для конкретного столбца, но для алгоритма это понятно далеко не всегда. Степень возможного отрицательного воздействия контрольных значений на результаты классификации зависит от конкретного используемого алгоритма машинного обучения.

Решения: отсутствующие (потерянные) данные

Рассмотрим вариант устранения описанной выше проблемы на конкретном примере[1] и поэкспериментируем с некоторыми различными решениями. Набором данных для этого примера является база данных, содержащая записи о 1470 сотрудниках некоторой организации в прошлом и настоящем. В этом наборе данных, представленном в табл. 7.1, имеется четыре столбца: TotalWorkingYears (общий рабочий стаж в годах), MonthlyIncome (месячный доход), Overtime (сверхурочная работа) и DailyRate (ставка в день). Метка Label указывает, уволился ли сотрудник из организации (значение 0 – сотрудник продолжает работать).

С помощью этого набора данных мы пытаемся сделать предположение о том, что сотрудник уволился (или, возможно, уволился) на основе других четырех признаков. Признак Overtime бинарный, остальные три признака числовые. Выполним обработку этого набора данных и попытаемся классифицировать его, используя классификатор на основе дерева решений, который уже знаком нам

[1] Полный код этого примера можно найти в Python Jupyter notebook *chapter7/missing-values-imputer.ipynb* (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter7/missing-values-imputer.ipynb**) в репозитории кода для данной книги.

по предыдущим главам. Сначала определим вспомогательную функцию, которая формирует модель и возвращает ее точность на тестовом наборе:

```
def build_model(dataset, test_size=0.3, random_state=17):
    # Разделение данных на тренировочный и тестовый наборы
    X_train, X_test, y_train, y_test = train_test_split(
        dataset.drop('Label', axis=1), dataset.Label,
        test_size=test_size, random_state=random_state)

    # Подгонка классификатора на основе дерева решений
    clf = DecisionTreeClassifier(
        random_state=random_state).fit(X_train, y_train)

    # Вычисление точности
    y_pred = clf.predict(X_test)
    return accuracy_score(y_test, y_pred)
```

Теперь попробуем создать модель на всем наборе данных:

```
# Чтение данных в объект DataFrame
df = pd.read_csv('employee_attrition_missing.csv')
build_model(df)
```

Здесь библиотека scikit-learn выдает сообщение об ошибке:

```
> ValueError: Input contains NaN, infinity or a value too large for
      dtype('float32').
```

Похоже, что некоторые значения в исследуемом наборе данных пропущены. Рассмотрим подробнее содержимое объекта DataFrame:

```
df.head()
```

Таблица 7.1. Экземпляры строк, выборочно взятые из набора данных по убыли персонала

	TotalWorkingYears	MonthlyIncome	Overtime	DailyRate	Label
0	NaN	6725	0	498.0	0
1	12.0	2782	0	NaN	0
2	9.0	2468	0	NaN	0
3	8.0	5003	0	549.0	0
4	12.0	8578	0	NaN	0

Из табл. 7.1 видно, что в нескольких строках содержатся значения NaN[1] для столбцов TotalWorkingYears и DailyRate.

Существует пять методов, которые можно использовать для устранения проблемы пропущенных значений в наборах данных.

1. Исключить строки с любыми пропущенными значениями (без замены).
2. Исключить столбцы, содержащие пропущенные значения.
3. Заполнить пропущенные значения, собрав больший объем данных.
4. Заменить пропущенные значения нулями или некоторыми другими значениями-«индикаторами».
5. Заменить пропущенные значения (условно-корректными значениями – импутация).

[1] NaN (Not a Number) – не числовое значение.

Метод 1 работает, если пропущенные значения содержатся в малом количестве строк и точки данных представлены в изобилии. Например, если только в 1 % экземпляров обнаружены пропущенные данные, то вполне допустимо удалить эти строки. Метод 2 можно применить, если признаки, для которых некоторые строки содержат пропущенные значения, не являются чрезвычайно значимыми признаками для процесса обучения. Например, если только в столбце Age (возраст) обнаружены пропущенные значения, а признак возраста не оказывает сколько-нибудь существенного влияния на работу алгоритма обучения (т. е. удаление этого признака не приведет к значительному снижению точности прогнозов), то приемлемым вариантом будет полное исключение столбца Age из процесса обучения. Методы 1 и 2 просты, но большинство операторов редко встречается с ситуацией, в которой они располагают достаточным объемом данных или признаков, позволяющим исключать строки или столбцы без снижения эффективности.

Определим, какая часть экземпляров в нашем наборе данных содержит пропущенные значения. Исключить все строки, содержащие значения NaN, можно с помощью функции `pandas.DataFrame.dropna()` (**https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.dropna.html**):

```
num_orig_rows = len(df)
num_full_rows = len(df.dropna())

(num_orig_rows - num_full_rows)/float(num_orig_rows)

> 0.5653061224489796
```

Более половины строк содержат, как минимум, одно пропущенное значение, и в двух из четырех столбцов обнаружены пропущенные значения – удаление строк или столбцов не приемлемо. Все же рассмотрим, как методы 1 и 2 работают с нашим набором данных:

```
df_droprows = df.dropna()
build_model(df_droprows)

> 0.75520833333333337

df_dropcols = df[['MonthlyIncome','Overtime','Label']]
build_model(df_dropcols)

> 0.77324263038548757
```

Исключение строк с пропущенными значениями дает точность классификации 75,5 %, а при удалении столбцов с пропущенными значениями получаем точность 77,3 %. Попробуем улучшить эти результаты.

Методы 3 и 4 пытаются заполнить пропуски данных вместо удаления «неправильных» строк (и столбцов). Метод 3 дает данные наивысшего качества, но чаще всего он оказывается невыполнимым и слишком дорогим. В рассматриваемом здесь примере потребуется слишком много затрат на розыск каждого сотрудника, только для того чтобы заполнить пустые графы таблицы. Также невозможно сгенерировать больший объем данных до тех пор, пока на работу в компании не будет принято достаточно много новых сотрудников.

Попробуем применить метод 4, заполняя все пропуски в данных контрольным значением –1 (так как все данные в наборе неотрицательные, значение –1 является хорошим индикатором пропущенных данных):

```
# Замена всех пропущенных значений NaN на -1
df_sentinel = df.fillna(value=-1)
build_model(df_sentinel)
```

```
> 0.75283446712018143
```

Применение этого метода дает точность классификации 75,3 % – это хуже, чем простое удаление строк или столбцов с пропущенными данными. Здесь мы наблюдаем на практике опасность примитивной подмены значений без учета того, что могут означать подставляемые числа.

Теперь сравним полученные выше результаты с результатом, который позволяет получить метод 5. Импутация (imputation) обозначает операцию замены пропущенных значений разумно выбранными значениями, которые сводят к минимуму воздействие этой замены (т. е. новых вводимых значений вместо пропущенных) на распределение в исследуемом наборе данных. Другими словами, необходимо сделать так, чтобы новые значения, заполняющие пропуски в данных, не оказывали существенного отрицательного воздействия на исследуемый набор (не загрязняли его слишком сильно). Наилучшим способом выбора значения для замены пропущенных данных обычно является использование среднего или срединного значения или наиболее часто встречающегося значения (моды) в столбце. Выбор конкретного способа зависит от природы исследуемого набора данных. Если набор данных содержит много промахов (выбросов), например если 99 % значений DailyRate меньше 1000, а 1 % больше 100 000, то замена пропущенных данных на среднее значение не подходит.

Библиотека scikit-learn предоставляет удобную утилиту для импутации пропущенных значений: sklearn.preprocessing.Imputer (**http://lijiancheng0614.github.io/scikit-learn/modules/generated/sklearn.preprocessing.Imputer.html**). Воспользуемся этой утилитой для замены всех пропущенных значений на соответствующим образом выбранные средние значения для каждого столбца с пропущенными данными:

```
from sklearn.preprocessing import Imputer

imp = Imputer(missing_values='NaN', strategy='mean', axis=0)

# Создание нового объекта DataFrame с набором данных, преобразованным утилитой Imputer
df_imputed = pd.DataFrame(imp.fit_transform(df),
                          columns=['TotalWorkingYears', 'MonthlyIncome',
                                   'OverTime', 'DailyRate', 'Label'])
build_model(df_imputed)
```

```
> 0.79365079365079361
```

Точность классификации сразу же увеличилась до 79,4 %. Вполне обоснованно можно предположить, что чаще всего импутация является наилучшим вариантом устранения проблемы пропущенных значений.

Качество модели

Тренированные модели формируют ядро «разума» систем машинного обучения. Но без защитных мер и средств, обеспечивающих качество этих моделей, генерируемые ими результаты будут неоптимальными. Модели могут существовать

в различных формах в зависимости от конкретного используемого алгоритма машинного обучения, но по существу это структуры данных, содержащие параметры, обучение которым происходит на этапе тренировки алгоритма. Например, тренируемая модель дерева решений содержит все разветвления и значения в каждом узле, в то время как тренируемая модель классификации методом k-ближайших соседей (k-NN) (в простейшей реализации[1] или дерево гиперсфер (ball tree; оно же дерево метрик – metric tree)[2]) в действительности представляет собой весь тренировочный набор данных.

Качество модели важно не только на этапе начальной тренировки и развертывания. Качеству модели необходимо уделять постоянное внимание и во время развивающегося противостояния вашей системы и любой вредоносной деятельности – регулярное сопровождение и повторные оценки гарантируют, что система не будет ухудшаться со временем.

Проблема: оптимизация гиперпараметров

Гиперпараметры – это параметры алгоритма машинного обучения, которым он не обучается во время обычного тренировочного процесса. Рассмотрим некоторые примеры точно настраиваемых гиперпараметров для классификатора `DecisionTreeClassifier` (**https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html**) из библиотеки scikit-learn:

```
from sklearn import tree
classifier = tree.DecisionTreeClassifier(max_depth=12,
                                         min_samples_leaf=3,
                                         max_features='log2')
```

В конструкторе этого классификатора мы определяем параметр `max_depth`, ограничивающий рост дерева 12 уровнями. Если этот параметр не определен, то поведение по умолчанию в данной реализации предусматривает разделение узлов до тех пор, пока все листы не станут «чистыми» (т. е. содержат экземпляры, принадлежащие единственному классу), или, если определен параметр `min_samples_split`, рост дерева останавливается, когда все узлы-листы содержат меньше экземпляров, чем задано в параметре `min_sample_split`. Также определяется параметр `min_samples_leaf=3`, согласно которому алгоритм должен обеспечить минимум три экземпляра данных в каждом узле-листе. Для параметра `max_features` задано значение `log2`, показывающее, что максимальное число признаков, которое должен рассматривать классификатор при поиске налучшего разделения узла, равно логарифму по основанию 2 от общего количества признаков в данных. Если значение `max_features` не задано, то по умолчанию принимается количество всех признаков. Полный список точно настраиваемых гиперпараметров для любого

---

[1] Большинство реализаций алгоритмов k-NN в действительности не хранит весь тренировочный набор данных как модель. Для повышения эффективности по времени прогнозирования реализации k-NN чаще всего используют такие структуры данных, как k-d деревья. См. *J. L. Bentley*. Multidimensional Binary Search Trees Used for Associative Searching. Communications of the ACM 18:9 (1975): 509.

[2] *A. M. Kibriya and E. Frank*. An Empirical Comparison of Exact Nearest Neighbour Algorithms. Proceedings of the 11th European Conference on Principles and Practice of Knowledge Discovery in Databases (2007): 140–151.

классификатора можно найти в соответствующей документации. Этот список может быть весьма длинным.

Значения гиперпараметров обычно необходимо выбрать перед началом этапа тренировки. Но как можно заранее узнать, какое значение следует установить для скорости тренировки? Или сколько скрытых слоев в глубокой нейронной сети позволят получить наилучшие результаты? А какое значение k использовать при кластеризации методом k-средних? Эти выглядящие случайными решения могут оказать значительное воздействие на эффективность модели. Начинающие практики обычно пытаются избежать сложностей, оставляя без изменений значения по умолчанию, предлагаемые используемой библиотекой машинного обучения. Многие тщательно разработанные библиотеки машинного обучения (включая scikit-learn) действительно предлагают разумно выбранные значения по умолчанию, подходящие для большинства реальных ситуаций. Но установить набор гиперпараметров, которые были бы оптимальными для всех случаев, невозможно. Каждый инженер машинного обучения обязан понимать используемые алгоритмы настолько хорошо, чтобы подбирать оптимальное сочетание гиперпараметров для каждой конкретной задачи. Поскольку совокупность параметров чрезвычайно велика, процесс подбора значений для них может быть дорогостоящим и медленным, даже для экспертов по машинному обучению.

Решения: оптимизация гиперпараметров

Гиперпараметры представляют собой чрезвычайно чувствительный компонент систем машинного обучения, так как на их оптимальность могут повлиять даже небольшие изменения во входных данных или в других частях системы. Эту проблему можно решить простейшим способом «грубой силы», т. е. тренировкой различных моделей с использованием всех возможных сочетаний значений гиперпараметров алгоритма, а затем выбрать тот набор гиперпараметров, при котором модель выдает самые лучшие результаты и демонстрирует самую высокую эффективность.

Оптимизация гиперпараметров чаще всего выполняется с использованием методики, именуемой поиск по решетке (grid search), или вариация параметров, с исчерпывающим охватом всего пространства гиперпараметров алгоритма машинного обучения. Используя метрику сравнения для оценки того, насколько хорошо каждый классификатор выполняет свою работу при различных сочетаниях значений гиперпараметров, можно найти оптимальную конфигурацию. Даже с учетом того, что эта операция связана с большим объемом вычислений, ее можно без труда распараллелить, поскольку каждый вариант конфигурации гиперпараметров может быть вычислен независимо от других вариантов. Библиотека scikit-learn предоставляет класс sklearn.model_selection.GridSearchCV (**https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html**), который обеспечивает реализацию этой возможности.

Рассмотрим небольшой пример практического применения метода опорных векторов для решения задачи цифровой классификации, но вместо наиболее часто используемого набора данных MNIST будем работать с набором данных меньшего размера и менее требовательного к вычислительным ресурсам, включенного в комплект наборов данных `digits` библиотеки scikit-learns (**https://scikit-learn.org/stable/auto_examples/datasets/plot_digits_last_image.html**), сформированный на

основе цифрового набора данных Pen-Based Recognition of Handwritten Digits Data Set (**https://archive.ics.uci.edu/ml/datasets/Pen-Based+Recognition+of+Handwritten+Digits**). Перед началом процедуры оптимизации гиперпараметров настоятельно рекомендуется установить базовый уровень производительности (эффективности) при значениях гиперпараметров, заданных по умолчанию:

```
from sklearn import svm, metrics
from sklearn.model_selection import train_test_split
from sklearn.datasets import fetch_mldata, load_digits

# Чтение набора данных и разделение его на тестовый и тренировочный наборы
digits = load_digits()
n_samples = len(digits.images)
data = digits.images.reshape((n_samples, -1))
X_train, X_test, y_train, y_test = train_test_split(
    data, digits.target, test_size=0.3, random_state=0)

# Тренировка SVC-классификатора, затем прогнозирование и оценка точности
classifier = svm.SVC()
classifier.fit(X_train, y_train)
predicted = classifier.predict(X_test)
print("Accuracy:  %.3f"  % metrics.accuracy_score(y_test, predicted))

> Accuracy: 0.472
```

Точность 47,2 % – это очень плохой результат. Попытаемся улучшить этот результат с помощью тщательной настройки гиперпараметров:

```
from sklearn.svm import SVC
from sklearn.model_selection import GridSearchCV

# Определение словаря, содержащего все значения гиперпараметров для пробных вычислений
hyperparam_grid = {
    'kernel': ('linear', 'rbf'),
    'gamma': [0.00001, 0.0001, 0.001, 0.01, 0.1, 1],
    'C': [1, 3, 5, 7, 9]
}

# Выполнение поиска по решетке с очередным пробным набором гиперпараметров и классификатором
classifier = GridSearchCV(svc, hyperparam_grid)
classifier.fit(X_train, y_train)
```

Словарь `hyperparam_grid`, передаваемый в конструктор класса `GridSearchCV` вместе с объектом `svc`, вычисляющим оценку, содержит все значения гиперпараметров, которые передаются в алгоритм поиска по решетке. Затем алгоритм создает 60 моделей, по одной для каждого возможного сочетания гиперпараметров, и выбирает самую лучшую:

```
print('Best Kernel:  %s'  % classifier.best_estimator_.kernel)
print('Best Gamma:  %s'  % classifier.best_estimator_.gamma)
print('Best C:  %s'  % classifier.best_estimator_.C)

> Best Kernel: rbf
> Best Gamma:  0.001
> Best C:      3
```

Значениями по умолчанию, предоставляемыми классом `sklearn.svm.SVC`, являются `kernel='rbf'`, `gamma=1/n_features` (для рассматриваемого набора данных `n_features=64`, поэтому `gamma=0.015625`) и `C=1`. Отметим, что `gamma` и `C`, предлагаемые `GridSearchCV`, отличаются от значений по умолчанию. Рассмотрим, как эта конфигурация работает на тестовом наборе данных:

```
predicted = classifier.predict(X_test)
print("Accuracy:  %.3f"  % metrics.accuracy_score(y_test, predicted))
```

```
> Accuracy: 0.991
```

Точность значительно улучшилась. Метод опорных векторов весьма чувствителен к этим гиперпараметрам, особенно к коэффициенту ядра `gamma`, по причинам, которые здесь не рассматриваются.

Класс `GridSearchCV` может затратить некоторое время на обработку данных, так как тренирует отдельный SVC-классификатор для каждого сочетания гиперпараметров, подбираемого при поиске по решетке. Этот процесс может оказаться весьма дорогостоящим, особенно при обработке больших наборов данных. Библиотека scikit-learn предоставляет более эффективные алгоритмы оптимизации гиперпараметров, как, например, `sklearn.model_selection.RandomizedSearchCV` (**https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html**), который может возвращать результаты быстрее.

Даже для алгоритмов с небольшим количеством гиперпараметров метод поиска по решетке является способом, требующим слишком много времени и ресурсов для решения задачи из-за проблемы комбинаторного взрыва (combinatorial explosion). Приняв этот простейший алгоритм на нижнюю (базовую) границу эффективности, рассмотрим несколько способов оптимизации данного процесса.

1. Хорошее понимание алгоритма и его параметров. Хорошее понимание внутренней работы алгоритма и наличие опыта по его практической реализации способны успешно провести вас через весь итеративный процесс ручной оптимизации гиперпараметров и помочь избежать тупиковых ситуаций. Но даже если вы новичок в этой области, процесс точной настройки не должен выполняться вслепую. Визуализация результатов тренировки обычно подсказывает пути улучшения гиперпараметров в определенном направлении и/или по величине. Рассмотрим классический пример нейронной сети для классификации цифр (от 0 до 9) из набора данных MNIST[1], состоящего из отдельных рукописных цифр. Для этой задачи используется модель полностью связанной пятиуровневой нейронной сети, реализованной в TensorFlow. Применяя инструментальное средство визуализации TensorBoard, включенное в стандартный дистрибутив TensorFlow, мы строим график потерь `cross_entropy`:

   ```
   cross_entropy = tf.nn.softmax_cross_entropy_with_logits(
       logits=Ylogits, labels=Y_)
   cross_entropy = tf.reduce_mean(cross_entropy)*100
   tf.summary.scalar('cross_entropy', cross_entropy)
   ```

На рис. 7.1 показан результат.

[1] *Yann LeCun, Corinna Cortes and Christopher Burges*. The MNIST Database of Handwritten Digits (**http://yann.lecun.com/exdb/mnist/**) (1998).

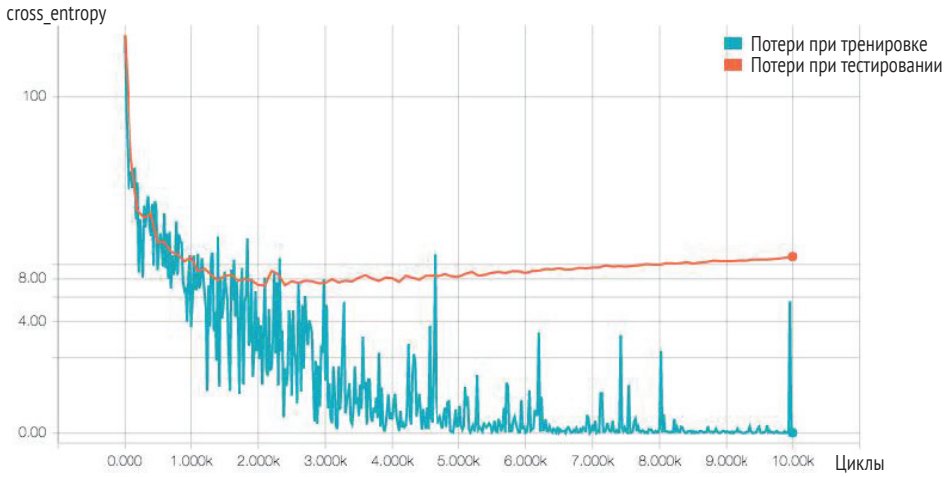


Рис. 7.1 ❖ Скалярный график TensorBoard процесса тренировки и тестирования потерь cross_entropy (в логарифмическом масштабе)

Наблюдая за процессом тренировки и тестирования потерь cross_entropy на протяжении 10 000 циклов на рис. 7.1, следует отметить любопытное разветвление этих двух трендов. Тренировка выполняется на выборке из 55 000 цифр, а для тестирования используется статический набор из 10 000 экземпляров цифр. После каждого цикла потери `cross_entropy` вычисляются отдельно для тренировочного набора данных (используемого для тренировки нейросети) и тестового набора данных (который недоступен для нейросети на этапе тренировки). Как и ожидалось, потери при тренировке снижаются до нуля при увеличении количества циклов тренировки, свидетельствуя о том, что нейросеть постепенно улучшает качество своей работы с увеличением времени обучения. Потери при тестировании сначала соответствуют шаблону, аналогичному схеме потерь при тренировке, но приблизительно после 2000 циклов число потерь начинает возрастать. Обычно это является явным признаком того, что происходит переподгонка нейросети на тренировочных данных. Если вы когда-либо имели дело с нейронными сетями, то знаете, что метод исключения (dropout)[1] является действенным способом регуляризации и устраняет проблему переподгонки. Применение метода исключения к рассматривамой здесь нейросети устранит тенденцию роста потерь при тестировании. Итеративно подбирая различные значения для метода исключения, применяемого к рассматриваемой нейросети, можно воспользоваться этим же графиком для поиска значений гиперпараметров, которые позволят устранить переподгонку без существенной потери точности.

2. Имитация похожих моделей. Другим часто используемым способом решения проблемы «холодного запуска» гиперпараметров является исследование ранее выполненных подобных работ в этой области. Даже если

[1] *Nitish Srivastava et al.* Dropout: A Simple Way to Prevent Neural Networks from Overfitting. Journal of Machine Learning Research 15 (2014): 1929–1958.

задача не вполне совпадает по своей природе с вашей, копирование гиперпараметров из другой работы при полном понимании причин их выбора, возможно, позволит сэкономить немало времени, поскольку кто-то уже выполнил работу по решению похожей задачи. Например, может потребоваться 10 повторений эксперимента для определения оптимального коэффициента исключения 0,75 при использовании нейросети для классификации MNIST, но поиск значений, применяемых для решения задачи MNIST в ранее опубликованных работах с использованием похожей нейронной сети, может сократить время оптимизации гиперпараметров в вашем случае.

3. Не начинать точную настройку параметров слишком рано. В условиях ограничения ресурсов не следует беспокоиться о гиперпараметрах. Единственный случай, когда следует вплотную заняться параметрами, – если возникли подозрения, что они могут являться причиной проблем, возникших в классификаторе. Необходимо начать с простейших конфигураций и внимательно наблюдать за возможными улучшениями при постепенных изменениях – это наилучшая практическая методика в общем случае.

AutoML (**https://www.automl.org/**) – область исследований, ориентированных на автоматизацию процесса тренировки и точной настройки систем машинного обучения, в том числе и процесса оптимизации гиперпараметров. По существу, инструментальные средства AutoML способны выбрать наилучший алгоритм для задачи, сформировать оптимальную архитектуру поиска для глубоких нейронных сетей и проанализировать важность влияния гиперпараметров на результат прогнозирования. Несмотря на то что это направление пока еще находится на этапе теоретических исследований, AutoML, несомненно, является областью, заслуживающей пристального внимания.

Дополнительные функции: циклы обратной связи, A/B-тестирование моделей

Поскольку системы машинного обучения для обеспечения безопасности обладают весьма низкой устойчивостью к неточностям, каждый бит обратной связи с пользователем должен быть принят во внимание для улучшения эффективности системы, насколько это возможно. Например, система выявления аномалий, генерирующая слишком много ложноположительных предупреждений для персонала по обеспечению безопасности, должна быть улучшена посредством присваивания корректных меток экспертами-людьми на этапе классификации сигналов-предупреждений для повторной тренировки и усовершенствования модели.

Кроме того, работающие долговременно системы машинного обучения нередко попадают в затруднительное положение из-за изменения концепции (concept drift) (или деградации модели (model rot)), когда модель, которая изначально выдавала правильные результаты, ухудшает свою работу со временем. Часто это является следствием изменения свойств входных данных из-за внешних воздействий, поэтому системы машинного обучения должны обладать достаточной гибкостью, чтобы приспосабливаться к таким изменениям.

Первым шагом к обеспечению гибкости системы является выявление деградации модели, прежде чем она нанесет ущерб системе, генерируя неправильные

или бессмысленные результаты. Циклы обратной связи представляют собой хороший способ не только для выявления ухудшения работы модели, но также для сбора помеченных тренировочных данных для непрерывного усовершенствования системы. На рис. 7.2 показана простая система выявления аномалий с интегрированным в нее циклом обратной связи.

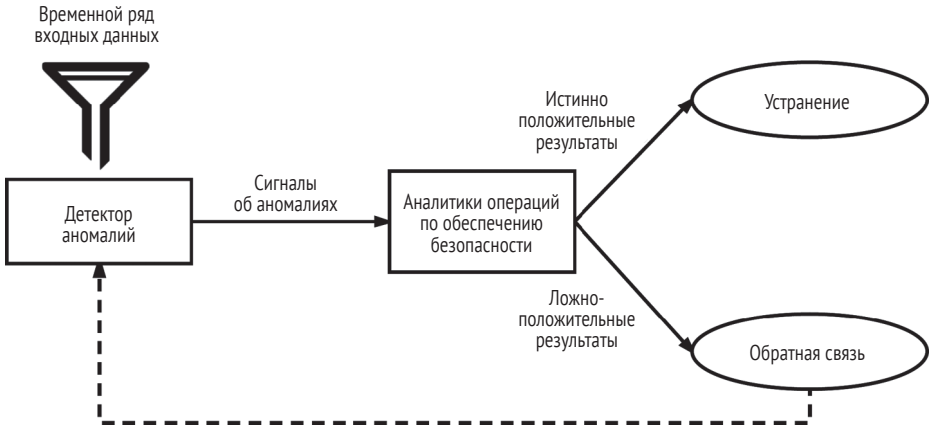


Рис. 7.2 ❖ Система выявления аномалий с циклом обратной связи

Штриховая линия на рис. 7.2 обозначает канал передачи информации, который аналитики операций по обеспечению безопасности могут использовать для создания экспертной обратной связи с системой. Ложноположительные результаты, генерируемые детектором, помечаются экспертами-людьми, которые затем могут воспользоваться каналом обратной связи для сообщения системе о совершенной ею ошибке. Далее система может преобразовать это сообщение обратной связи в точку данных с соответствующей меткой и использовать ее для будущей тренировки. Циклы обратной связи чрезвычайно ценны, потому что создаваемые ими тренировочные экземпляры данных с метками представляют самые трудноопределяемые варианты прогнозов, которые должна выполнять система и которые могут помочь системе избавиться от подобных ошибок в будущем. Отметим, что повторное обучение с циклом обратной связи может привести к переподгонке, поэтому интеграция цикла обратной связи должна быть выполнена с соответствующей процедурой регуляризации. Кроме того, циклы обратной связи могут создавать потенциальную угрозу для системы безопасности, если аналитики операций по обеспечению безопасности не внушают абсолютного доверия или если система каким-либо образом взломана с целью внедрения вредоносной обратной связи. Результатом этого будет заражение модели извне, например атаки типа ложный маневр (red herring), из-за которых модель обучается на данных с намеренно искаженными метками и ее эффективность быстро снижается. В главе 8 будут рассматриваться стратегии снижения ущерба в ситуациях, в которых невозможно обеспечить достаточный уровень доверия внутри системы.

Обучение с подкреплением и активное обучение

Существуют два типа систем машинного обучения, которые тесно связаны с методикой обеспечения онлайновой обратной связи.

Обучение с подкреплением (reinforcement learning – RL) – это метод машинного обучения, при котором модель тренируется с помощью марковских процессов и цикла обратной связи. Алгоритмы обучения с подкреплением пытаются найти баланс между стохастическим исследованием (exploration) (для получения знаний, которыми модель не обладает) и реальным использованием знаний (exploitation) (для подкрепления ранее полученных знаний). При помощи поощрения модели, когда получен положительный сигнал обратной связи, и наказании модели при получении отрицательного сигнала обратной связи модели обучения с подкреплением тренируются по методу, совершенно отличающемуся от методов обучения с учителем, где «обратная связь» предоставляется алгоритмам с самого начала в форме меток.

Активное обучение (active learning) – это особый тип обучения с частичным привлечением учителя, при котором тренируемая модель классификатора должна выбирать точки данных, для которых она с меньшей уверенностью выполняет прогнозирование, и предлагает экспертам-людям присвоить метки этим «сомнительным» данным. Люди присваивают правильные метки через цикл обратной связи, после чего алгоритм использует их для тренировки и усовершенствования модели. Активное обучение полезно в области обеспечения безопасности, потому что эта методика хорошо подходит для ситуаций, в которых наблюдается недостаток наборов данных с правильными метками в контексте обеспечения безопасности. Существуют различные стратегии выбора элементов для передачи их на рассмотрение экспертам-людям[1], но здесь они не описываются. Если вы планируете применение метода активного обучения для улучшения точности модели, то можно порекомендовать тщательное изучение специальной литературы по этой теме.

A/B-тестирование (A/B-testing), или тестирование с разделением (split testing), – это рандомизированный управляемый эксперимент, предназначенный для понимания того, как различные варианты системы влияют на метрики. В настоящее время самые крупные веб-сайты запускают одновременно сотни и даже тысячи A/B-тестов как различные целевые группы для оптимизации множества метрик. Стандартная процедура A/B-тестирования – случайное разделение совокупности пользователей на две группы, A и B, и предоставление каждой группе отдельного варианта тестируемой системы (например, классификатора спама). Процедура определения оценки результатов эксперимента состоит из сбора данных по тестируемой метрике от каждой группы и выполнения статистического теста (обычно t-теста или определения критерия хи-квадрат) для вычисления значимости статистического различия между метриками двух групп.

Главная трудность при проведении A/B-тестирования – определение объема трафика, проходящего через новую систему (A, или исследуемая группа), и объема трафика, проходящего через старую систему (B, или контрольная группа). Эта задача является одной из вариаций задачи многорукого, или N-рукого, бандита

[1] *Burr Settles*. Active Learning Literature Survey. Computer Sciences Technical Report 1648, University of Wisconsin–Madison (2010).

(multi-armed/N-armed bandit) в теории вероятностей, при решении которой необходимо сохранять баланс между исследованием с целью получения новых знаний (exploration) и практическим использованием ранее полученных знаний (exploitation). Необходима возможность узнать как можно больше о новой системе, выполняя маршрутизацию и мониторинг трафика для нее (другими словами, необходимо получить максимум статистической мощности для проведения теста), но при этом не приемлем риск общей деградации метрик из-за того, что эффективность (производительность) системы A может быть хуже, чем эффективность существующей системы B. Одним из алгоритмов, позволяющим решить такую задачу, является выборка Томпсона (Thompson sampling), которая при маршрутизации в каждом варианте включает объем трафика, пропорциональный вероятности получения улучшенного результата в будущем на основе ранее собранных данных. Контекстные многорукие бандиты (contextual multi-armed bandits)[1] используют этот подход в расширенной форме, а также привносят дополнительные факторы среды в этот процесс принятия решений.

При использовании систем машинного обучения необходимо всегда проверять и сравнивать новые поколения моделей с моделями, находящимися в реальной эксплуатации, с помощью A/B-тестирования. Применяя этот тест, необходимо иметь четко определенную метрику, на оптимизацию которой ориентирован тест. Например, такой метрикой для классификатора спама при A/B-тестировании может быть количество спам-сообщений электронной почты, собранных в почтовом ящике входящих сообщений пользователя. Эта метрика измеряется посредством обратной связи с пользователем или с помощью выборки данных и присваивания им меток.

A/B-тестирование чрезвычайно важно для систем машинного обучения, поскольку постепенные обновления долго работающих моделей (например, повторные тренировки) могут не дать наилучших ожидаемых результатов. Возможность экспериментирования с новыми моделями и эмпирическое определение условий достижения наивысшей эффективности придает системам машинного обучения гибкость, требуемую для приспособления к изменениям в области данных и алгоритмов.

Но в потенциально опасных средах A/B-тесты необходимо проводить с величайшей осторожностью. Лежащая в основе A/B-тестирования статистическая теория предполагает, что распределение входных данных одинаково в сегментах A и B. Но тот факт, что вы передаете даже небольшую часть трафика в новую модель, может стать причиной изменения поведения атакующего противника. В этом случае предположение перед A/B-тестированием не оправдывается, и получаемые статистические данные не будут иметь смысла. Кроме того, даже если вредоносный трафик разделяется между сегментами, тот факт, что трафик теперь обрабатывается по-другому, может заставить атакующего изменить поведение или даже скрыться, и действительно необходимые вам метрики (объем рассылаемого спама) могут не показывать статистически значимого различия в A/B-тесте, даже если новая модель эффективна. Если вы начинаете блокировать 50 % вредоносного трафика в новой модели, то противник просто удвоит скорость запросов, а ваша замечательная модель в итоге не изменит значения метрик.

---

[1] *Tyler Lu, Dávid Pál and Martin, Pál*. Contextual Multi-Armed Bandits. Journal of Machine Learning Research Proceedings Track 9 (2010): 485–492.

Воспроизводимые и объяснимые результаты

Иногда недостаточно просто получить правильный ответ. Во многих случаях результаты прогнозов должны быть воспроизводимыми повторно для проведения аудитов, отладки и апелляций. Если онлайновая модель защиты учетных записей помечает учетные записи пользователя как подозрительные, то необходимо обоснование такого решения, принятого на основе прогнозирования. Системы должны обладать способностью предсказуемого повторного воспроизведения результатов и избавляться от любых эффектов стохастического непостоянства в цепочке последовательного принятия решений.

Системы машинного обучения часто оцениваются по одной метрике: точность прогнозов. Но почти всегда в реальных эксплуатационных средах существуют более важные факторы, которые влияют на успешность и адаптируемость систем машинного обучения, особенно в сфере обеспечения безопасности. Взаимоотношения человека и машины осложняются недоверием, поэтому система, которая не способна убедить человека в том, что она принимает верные решения (особенно если такая система еще и неудобна в использовании), будет быстро отвергнута. Более того, системы машинного обучения для обеспечения безопасности чаще всего размещаются в зоне непосредственных действий с реальными последствиями (которые могут обойтись слишком дорого). Если классификатор вредоносных DNS выявляет подозрительный DNS-запрос, сделанный с компьютера пользователя, то вполне разумной и простой стратегией может быть блокировка такого запроса. Но подобное ответное действие вызывает нарушение рабочего потока пользователя, что в большинстве случаев приводит к действиям пользователя, требующим дополнительных затрат, например вызов сотрудников службы технической поддержки. В тех случаях, когда пользователь не может быть уверен в том, что его действие не имеет отрицательных последствий, он может даже попытаться найти способы обхода системы выявления (часто такие попытки бывают успешными, поскольку редко удается охватить все возможные поверхности риска).

Кроме формирования доверительных отношений между человеком и машиной, возможно, даже более важным эффектом от воспроизводимых и объясняемых результатов является то, что обслуживающий персонал системы и инженеры по машинному обучению получают возможность анализировать, оценивать и отлаживать такие системы. Без внутреннего исследования усовершенствование систем машинного обучения становится чрезвычайно трудной задачей.

Повторное воспроизведение (repeatability) прогнозов систем машинного обучения – это простая концепция: предполагая постоянное изменение предварительных условий в статистической системе (из-за непрерывной адаптации, ручного вмешательства в процесс развития и т. д.), мы должны иметь возможность повторного воспроизведения любого решения, принятого системой в любой разумно взятый момент времени в ее хронологии. Например, если непрерывно адаптируемый классификатор вредоносного ПО ранее помечал бинарные файлы как корректные, но в какой-то момент вдруг решил, что они вредоносные, то чрезвычайно полезно иметь возможность повторного воспроизведения последних результатов и сравнения состояний системы (параметры/гиперпараметры) в различные моменты времени. Этого можно достичь, регулярно проверяя состояние системы в контрольных точках и сохраняя описание модели в надежном месте

с возможностью оперативного восстановления. Другим способом повторного воспроизведения результатов является фиксация в журнале параметров модели при каждом принятии решения системой.

Объяснимость результатов (explainability) систем машинного обучения представляет собой более сложную концепцию. Что означает для системы машинного обучения объяснимость ее результатов? Если вы задумаетесь над тем, насколько трудно объяснить каждое принятое вами решение другому человеку, то начнете понимать, что такое требование к системам машинного обучения не является абсолютно ясным и четким. Пока еще это весьма важная область теоретических исследований, привлекающая внимание академических кругов, представителей промышленности и правительства. В соответствии с документом агенства DARPA «целью программы Explainable AI (XAI) является создание комплекта новых или модернизированных методик машинного обучения, создающих объяснимые модели, которые в сочетании с эффективными методиками объяснения, доступными для понимания конечными пользователями, обладают соответствующим уровнем доверия и эффективно управляют новым развивающимся поколением систем искусственного интеллекта». В этой цитате точно определена долгосрочная цель, но существуют некоторые конкретные меры, которые можно предпринять для улучшения объясняемости результатов современных систем машинного обучения.

Объяснимость чрезвычайно важна для создания доверительных отношений с конкретной системой машинного обучения. Если система выявления мошенничества обнаруживает подозрительное событие, то последующим побочным эффектом, вероятнее всего, будет привлечение человека, который может подвергнуть сомнению правильность этого решения. Если сигнал направлен к аналитикам операций по обеспечению безопасности, то им потребуется ручная проверка действительности факта мошенничества. Если причины генерации сигнала не очевидны, то аналитики могут ошибочно пометить это событие как ложный сигнал, даже если система приняла действительно правильное решение.

В сущности, система является объяснимой, если она предоставляет информацию о принятии решения, достаточную для того, чтобы позволить пользователю получить объяснение этого решения. Людям доступен полный культурный и экспериментальный контекст, который позволяет получать объяснения решений в отдельных точках данных, но встроить этот контекст в машины, чтобы сделать его доступным для них, в настоящее время очень трудно. Например, «человеческое объяснение» причин признания бинарного файла вредносным может состоять в том, что этот файл установил на вашем компьютере кейлоггер, пытаясь похитить регистрационные данные для онлайновой учетной записи. Но в большинстве случаев пользователи не требуют такой подробной информации. Если подобная система способна объяснить, что решение принято потому, что было обнаружено нестандартное системное событие – непредусмотренное обращение к драйверу клавиатуры, и такое поведение на основе долговременных хронологических наблюдений однозначно связывается с действиями вредоносного ПО, то этого вполне достаточно, чтобы пользователь понял, почему система приняла защитные меры.

Но в некоторых случаях объяснимость и воспроизводимость результатов не имеют большого значения. Когда Netflix рекомендует вам для домашнего про-

смотра кинофильм, который вас не интересует, стоит ли действительно беспокоиться о причинах? Значимость строгой ответственности за прогнозы и рекомендации является функцией оценки важности и воздействий отдельных решений, принятых системой. Каждое решение системы машинного обучения, предназначенной для обеспечения безопасности, может иметь значительные последствия, поэтому объяснимость и воспроизводимость результатов важны при вводе таких систем в реальную производственную эксплуатацию.

Генерация объяснений с помощью LIME

Некоторые существующие в настоящее время методы решают проблему объяснимости путем поиска локализованных сегментов входных данных, которые вносят наибольший вклад в общий результат прогнозирования. Local Interpretable Model-Agnostic Explanations (LIME)[1] и Model Explanation System (MES) Тернера (Turner)[2] принадлежат к этому классу методов. LIME (**https://github.com/marcotcr/lime**) определяет объяснения как локальные линейные аппроксимации поведения модели машинного обучения: «Хотя модель в целом может быть очень сложной, проще аппроксимировать ее с помощью приблизительного конкретного экземпляра». С помощью многократно повторяемой активизации различных комбинаций локализованных сегментов входных данных и передачи их в модель с последующим сравнением результатов, полученных при включении или исключении определенных сегментов, LIME может генерировать линейные и локализованные объяснения решений классификатора. Попробуем применить LIME для примера мультиномиального наивного байесовского классификатора спама из главы 1 и получить некоторые объяснения, которые помогут понять процесс принятия решений этой системой[3]:

```
from sklearn.pipeline import make_pipeline
from lime.lime_text import LimeTextExplainer

# Определение class_names с позициями в списке, соответствующими
# конкретной метке, т. е. 'Spam' -> 0, 'Ham' -> 1
class_names = ['Spam', 'Ham']

# Создание конвейера sklearn, который сначала применяет
# объект-векторизатор (CountVectorizer) к исследуемому набору данных,
# затем передает его в экземпляр вычислителя оценки mnb (MultinomialNB)
c_mnb = make_pipeline(vectorizer, mnb)

# Инициализация объекта LimeTextExplainer
explainer_mnb = LimeTextExplainer(class_names=['Spam', 'Ham'])
```

---

[1] *Marco Tulio Ribeiro, Sameer Singh and Carlos Guestrin*. Why Should I Trust You?: Explaining the Predictions of Any Classifier. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (2016): 1135–1144.

[2] *Ryan Turner*. A Model Explanation System. Black Box Learning and Inference NIPS Workshop (2015).
Ryan Turner. A Model Explanation System: Latest Updates and Extensions. Proceedings of the 2016 ICML Workshop on Human Interpretability in Machine Learning (2016): 1–5.

[3] Полный код этого примера представлен в формате Python Jupyter notebook chapter7/lime-explainability-spam-fighting.ipynb (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter7/lime-explainability-spam-fighting.ipynb**) в репозитории кода для данной книги.

Теперь можно воспользоваться explainer_mnb для генерации объяснений по отдельным выборкам данных:

```
# Случайный выбор X_test[11121] как экземпляра для генерации описания для
idx = 11121

# Использование LIME для объяснения прогноза с использованием по меньшей мере
# 10 признаков (случайно выбранных) в описании
exp_mnb = explainer_mnb.explain_instance(
    X_test[idx], c_mnb.predict_proba, num_features=10)

# Вывод результатов прогнозирования
print('E-mail file:  %s'  % 'inmail.' + str(idx_test[idx]+1))
print('Probability(Spam) =  %.3f'  % c_mnb.predict_proba([X_test[idx]])[0,0])
print('True class:  %s'  % class_names[y_test[idx]])
```

```
> E-mail file: inmail.60232
> Probability(Spam) = 1.000
> True class: Spam
```

При внимательном рассмотрении выдержки из темы/содержимого сообщения inmail.60232 становится понятно, что это действительно спам:

```
Bachelor Degree in 4 weeks, Masters Degree in no more than 2 months. University
Degree OBTAIN A PROSPEROUS FUTURE, MONEY-EARNING POWER and
THE PRESTIGE THAT COMES WITH HAVING THE CAREER POSITION YOUVE
ALWAYS DREAMED OF. DIPLOMA FROM PRESTIGIOUS NON-ACCREDITED
UNVERSITIES BASED ON YOUR PRESENT KNOWLEDGE AND PROFESSIONAL
EXPERIENCE.If you qualify …
```

Можно продолжить исследования более подробно и проверить список признаков с весами, созданный генератором объяснений. Эти взвешенные признаки представляют линейную модель, которая аппроксимирует мультиноминальный наивный байесовский классификатор в локализованной области специально сформированной выборки данных:

```
exp_mnb.as_list()
```

```
> [(u'PROSPEROUS', -0.0004273209832636173),
   (u'HolidaysTue', -0.00042036070378471198),
   (u'DIPLOMA', -0.00041735867961910481),
   (u'Confidentiality', -0.00041301526556397427),
   (u'Degree', -0.00041140081539794645),
   (u'682', -0.0003778027616648757),
   (u'00', -0.00036797175961264029),
   (u'tests', 4.8654872568674994e-05),
   (u'books', -4.0641140958656903e-05),
   (u'47', 1.0821887948671182e-05)]
```

На рис. 7.3 эти же данные представлены в графическом виде.

На графике видно, что слова PROSPEROUS, HolidaysTue, DIPLOMA и т. п. отрицательно воздействуют на всю выборку, которая должна была бы определяться как корректная. Точнее, удаление слова PROSPEROUS из этой выборки привело бы к тому, что алгоритм мультиноминальной наивной байесовской классификации определил бы этот пример как спам с достоверностью, уменьшенной на

0,0427 %. Такое объяснение, сгенерированное LIME, позволяет конечному пользователю проверить компоненты, привлеченные алгоритмом машинного обучения для принятия конкретного решения. Аппроксимируя произвольные модели машинного обучения с локализованной и линейной заменой модели (описываемой линейными взвешенными признаками, как показано на рис. 7.3), LIME не требует привлечения какого-либо специального семейства моделей и может без затруднений применяться к любым существующим системам.

Рис. 7.3 ❖ Линейные взвешенные признаки, вносящие вклад в прогноз наивного байесовского классификатора

Эффективность

Многие системы машинного обучения, предназначенные для обеспечения безопасности, по своей природе существуют непосредственно в трафике, в потоке данных, при этом им приходится принимать мгновенные решения с постоянным риском потери работоспособности. Выявление аномалии через 15 минут после наступления события – это чаще всего слишком поздно. Системы, от которых требуется способность к адаптации в реальном времени, также должны соответствовать высокому уровню для эффективной реализации постоянно развивающегося процесса повторной тренировки.

При создании промышленных систем машинного обучения существует множество требований к эффективности, гораздо более строгих, чем требования к экспериментальным прототипам. В некоторых случаях задержки прогноза, выходящие из диапазона миллисекунд, могут привести к полному отказу всей системы. Более того, системы будут с высокой вероятностью выходить из строя при высоком уровне рабочей нагрузки, если при проектировании не была предусмотрена устойчивость к критическим сбоям и гибкая масштабируемость. Рассмотрим некоторые методы обеспечения минимальных задержек и высокой масштабируемости систем машинного обучения.

Цель: минимальные задержки, высокая масштабируемость

Машинное обучение, особенно при работе с большими наборами данных, представляет собой задачу, связанную с огромным объемом вычислений. Библиотека scikit-learn демонстрирует хорошие показатели эффективности и производительности по любой метрике, и все участники этого проекта постоянно вносят

в него усовершенствования. Тем не менее эффективность (производительность) может оставаться недостаточной, не соответствующей требованиям некоторых приложений. Для систем машинного обучения в сфере обеспечения безопасности в критических процессах принятия решений приспособляемость конечных пользователей к ответам с большим временем задержек может быть ограничена. В подобных случаях неплохим проектным решением часто становится исключение систем машинного обучения из основного процесса взаимодействия между пользователями и системой.

Система защиты должна принимать решения в асинхронном режиме, когда это возможно, а также должна обладать способностью устранять или смягчать угрозы в отдельном независимом процессе (пути). Например, веб-приложение с системой выявления вторжений (IDS), реализованной с применением машинного обучения, может непрерывно принимать входящие запросы. Такая система IDS должна в реальном времени принимать решения, является ли запрос угрозой. Веб-приложение может позволить передать запрос для выполнения, если не получен ответ от IDS в течение заданного ограниченного интервала времени, чтобы не создавать неудобства для работы пользователей из-за неприемлемого времени задержек в периоды перегрузки системы. Когда система выявления вторжений возвращает конечный результат и сообщает, что предыдущий разрешенный запрос был подозрительным объектом, она может активизировать в веб-приложении триггер, который позволит передать сообщение о принятом решении. После этого веб-приложение может выбрать одно из нескольких нейтрализующих действий, например немедленное запрещение всех дальнейших запросов, выполняемых этим пользователем.

Но такое проектное решение может оказаться неподходящим в некоторых случаях. Например, если единственный вредоносный запрос может привести к несанкционированному доступу к важным секретным данным, то атакующий уже достигнет своей цели к тому времени, когда IDS примет свое решение. Атакующие могут даже перегружать систему ложными запросами, чтобы замедлить работу IDS и расширить окно атаки. В подобных случаях рекомендуется наращивание ресурсов для оптимизации системы машинного обучения для минимизации задержек, особенно под максимальной нагрузкой. (Кроме того, возможно устранение подобной проблемы еще на этапе проектирования системы – нельзя допускать, чтобы единственный запрос стал причиной несанкционированного доступа к важным данным или серьезного повреждения системы.)

Оптимизация эффективности

Для ускорения работы приложений машинного обучения можно выполнить поиск узких мест в среде выполнения программ, подобрать более эффективные алгоритмы или применить параллельное выполнение. Рассмотрим подробнее все эти варианты[1]:

---

[1] Параллельное выполнение как методика оптимизации эффективности (производительности) более подробно рассматривается в следующем разделе «Горизонтальное масштабирование с помощью распределенных вычислительных программных сред».

- профилирование и оптимизация программной среды. Профилирование ПО представляет собой метод динамического анализа производительности программы. Такое инструментирование (instrumenting) ПО выполняется с помощью инструментального средства, которое называется профайлером (profiler). Обычно профайлер вставляет свои ловушки или контрольные точки в выполняемый код программы и выполняет углубленный анализ во время выполнения каждого отдельного компонента. Собранные данные позволяют оператору наблюдать на самом низком уровне за внутренними характеристиками производительности ПО и выявлять узкие места, снижающие эффективность выполнения. Профилирование является общеизвестной и широко применяемой частью процесса разработки ПО, а инструменты профилирования входят в стандартный набор разработчика и должны активно применяться инженерами машинного обучения, пытающимися оптимизировать алгоритмы и/или системы для промышленной эксплуатации.

  Алгоритмы ядра библиотеки scikit-learn чаще всего являются Cython-обертками вокруг других широко известных и тщательно проработанных библиотек поддержки машинного обучения, написанных на языке C или C++. Например, классы метода опорных векторов `SVM` в библиотеке scikit-learn в основном связаны с библиотекой LIBSVM[1], написанной на C++. Более того, процедура умножения матриц (которая является одной из наиболее часто выполняемых операций в алгоритмах машинного обучения) и другие векторные вычисления обычно выполняется с помощью библиотеки NumPy, которая использует собственный код и оптимизации низкого уровня для ускорения операций[2]. Узкие места, снижающие производительность, существуют всегда, поэтому профилирование является действенным способом их выявления, если низкая производительность становится проблемой в вашем приложении машинного обучения. Профайлер, интегрированный в среду IPython (**https://pynash.org/2013/03/06/timing-and-profiling/**), является неплохим выбором для начального уровня профилирования. При работе с крупными наборами данных и моделями, активно работающими с оперативной памятью, программа с большей вероятностью может столкнуться с ограничениями по использованию памяти, нежели с ограничениями вычислительных ресурсов. В таких случаях может помочь профайлер памяти memory_profiler (**https://pypi.org/project/memory-profiler/**), позволяющий находить инструкции или операции, некорректно работающие с памятью, и устранять проблемы.

---

[1] *Chih-Chung Chang and Chih-Jen Lin*. LIBSVM: A Library for Support Vector Machines. Transactions on Intelligent Systems and Technology 2:3 (2011).

[2] См. рекомендации по достижению наилучшей производительности с помощью NumPy (Getting the Best Performance out of NumPy – **https://ipython-books.github.io/45-understanding-the-internals-of-numpy-to-avoid-unnecessary-array-copying/**) из книги Сирилла Россанта (Cyrille Rossant – **https://cyrille.rossant.net/**) «IPython Interactive Computing and Visualization Cookbook» (издательство Packt).

Программные инструментальные средства оптимизированной линейной алгебры

Библиотеки scikit-learn и NumPy используют тщательно оптимизированные программные инструментальные средства линейной алгебры, такие как Basic Linear Algebra Subprograms (BLAS) (**http://www.netlib.org/blas/**) и Linear Algebra PACKage (LAPACK) (**http://www.netlib.org/lapack/**), если распространение ПО связано с этими библиотеками. Если при профилировании предполагается, что узкие места производительности находятся в подпрограммах умножения матриц, то следует знать, что используемый вами дистрибутив scikit-learn может быть скомпилирован без использования BLAS, поэтому, возможно, вызывается более медленная функция `numpy.dot()` (**https://docs.scipy.org/doc/numpy/reference/generated/numpy.dot.html**). Добавление приведенных ниже строк в приложение, использующее библиотеку scikit-learn, может помочь предупредить о недоступности пакета BLAS или о неоптимальном вызове функции `numpy.dot()`:

```
import warnings
from sklearn.exceptions import NonBLASDotWarning
warnings.simplefilter('always', NonBLASDotWarning)
```

Список средств оптимизации уровня программной среды, которые доступны для приложений машинного обучения, можно продолжать бесконечно, но здесь мы не будем этого делать. Такие оптимизации способны увеличить скорость выполнения алгоритмов в 2–5 раз, но это далеко не всегда позволяет улучшить эффективность всего приложения в целом;

- оптимизация алгоритмов. Оптимизация алгоритмов и выбор эффективных моделей часто позволяют добиться существенного улучшения производительности. Некоторая потеря точности при значительном повышении эффективности может стать приемлемым компромиссом в зависимости от общего контекста. Поскольку выбор модели является процессом, в высокой степени зависимым от контекста и приложения, нельзя сформулировать четко определенный набор правил, позволяющий мгновенно улучшить производительность и масштабируемость, выбирая конкретные алгоритмы из доступного набора. Тем не менее ниже приводится несколько рекомендаций, которые могут оказаться полезными при выборе оптимального алгоритма:
  - при небольшом количестве признаков выполняется меньший объем арифметических операций. Это может улучшить производительность. Применение методов снижения размерности для удаления незначительных признаков из набора данных может повысить эффективность;
  - модели на основе деревьев (например, деревья решений, случайные леса), вероятнее всего, будут демонстрировать весьма высокую эффективность процесса прогнозирования, так как каждый запрос взаимодействует только с небольшой частью пространства модели (один путь от корня к листу в каждом дереве). В зависимости от архитектуры и выбора

гиперпараметров прогнозы нейронной сети иногда могут выполняться быстрее, чем прогнозы случайных лесов[1];

- линейные модели быстрее тренируются и быстрее работают в эксплуатационном режиме. Тренировку линейных моделей можно выполнять в параллельном режиме с использованием широко известного алгоритма Alternanting Direction Method of Multipliers (ADMM)[2], который позволяет тренировать крупные линейные модели с хорошим масштабированием;
- методам опорных векторов (SVM) присущи хорошо известные проблемы масштабируемости. Это одно из наиболее медленно тренируемых семейств моделей. Кроме того, методы SVM отличаются интенсивным потреблением памяти. Простые линейные методы опорных векторов обычно являются единственным выбором для развертывания на крупных наборах данных. Но вычисления могут выполняться достаточно быстро, если проекции ядра не слишком сложны. Возможно (но сложно) распараллеливание процесса тренировки SVM[3];
- алгоритмы глубокого обучения (глубокие нейронные сети) медленно тренируются и весьма интенсивно потребляют ресурсы (обычно требуют выполнения по меньшей мере миллионов операций умножения матриц), но легко распараллеливаются на соответствующем оборудовании, например при использовании графических процессоров (GPU) и современных инструментальных программных сред, таких как TensorFlow, Torch или Caffe;
- аппроксимирующие алгоритмы поиска ближайших соседей, такие как k-d деревья (рассматриваемые в главе 2), способны значительно ускорить операции вычисления расстояний наибольшей близости между точками в крупных наборах данных. Кроме того, в целом эти алгоритмы очень быстро тренируются и обладают весьма высокой средней производительностью при ограниченном количестве ошибок. Другим аппроксимирующим методом поиска ближайших соседей является чувствительное к местоположению хеширование (LSH, рассматриваемое в главе 1).

Горизонтальное масштабирование с помощью распределенных вычислительных программных сред

Параллельное выполнение является главным принципом оптимизации эффективности (производительности). Распределяя набор, состоящий из 100 незави-

---

[1] Эту рекомендацию необходимо принимать с учетом многочисленных предупреждающих замечаний, например: размер модели, использование GPU вместо CPU и т. д.

[2] *Stephen Boyd et al*. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. Foundations and Trends in Machine Learning 3 (2011): 1–122.

[3] *Edward Y. Chang et al*. PSVM: Parallelizing Support Vector Machines on Distributed Computers. Proceedings of the 20th International Conference on Neural Information Processing Systems (2007) 257–264.

симых вычислительных операций, по 100 серверам, можно достичь сокращения времени обработки до 100 раз (без учета задержек на операции ввода/вывода и перемещения данных). На многих этапах процесса машинного обучения можно получить преимущества от распараллеливания, но многие наборы данных и алгоритмы невозможно «распределить вслепую», так как не всегда каждый отдельный элемент операции может быть независимым от других. Например, процесс тренировки классификатора на основе случайного леса распараллеливается с потрясающей легкостью, потому что каждое случайное дерево решений в таком лесу создано независимо от других и может получать индивидуальные запросы для генерации окончательного прогноза. Но другие алгоритмы (например, методы опорных векторов) не так легко поддаются распараллеливанию, поскольку требуют частой передачи глобальных сообщений (между узлами) на этапах тренировки и прогнозирования. Иногда это приводит к экспоненциальному росту издержек при увеличении степени распределенности. Здесь мы не будем углубляться в теорию параллельных вычислений и обработки данных, а вместо этого рассмотрим, как на практике получить преимущества от применения специализированных программных сред для горизонтального масштабирования систем машинного обучения наиболее эффективными и быстрыми способами.

Распределенные методы машинного обучения касаются не только процессов тренировки классификаторов или алгоритмов кластеризации на нескольких компьютерах. Библиотека scikit-learn предназначена для выполнения на одном узле, но существуют некоторые типы задач, при решении которых больше подходит парадигма распределенных вычислений. Например, при оптимизации гиперпараметров и выполнении операции поиска моделей (обсуждаемых выше, в разделе «Проблема: оптимизация гиперпараметров» текущей главы) создается большое количество симметричных задач, независимых друг от друга. Эти типы легко распараллеливаемых задач хорошо подходят для выполнения в распределенных программных средах MapReduce[1], таких как Apache Spark (**https://spark.apache.org/**). Spark – это платформа распределенных вычислений с открытым исходным кодом, интенсивно использующая архитектуры на основе оперативной памяти, ленивые вычисления и оптимизацию графов вычислений для настройки высокой производительности в программах MapReduce-стиля.

Пакет для языка Python spark-sklearn (**https://github.com/databricks/spark-sklearn**) интегрирует программную среду Spark в библиотеку scikit-learn с ориентацией на оптимизацию гиперпараметров. Несмотря на то что (на момент написания данной книги) набор функциональных возможностей scikit-learn, реализованных в spark-sklearn, был достаточно ограниченным, имеющиеся в наличии классы предоставляют возможность их подстановки в уже существующие приложения scikit-learn. Рассмотрим, как класс `spark_sklearn.GridSearchCV`[2] может помочь при выполнении операции подбора гиперпараметров для классификатора цифр на основе метода опорных векторов из раздела «Решения: оптимизация гиперпараметров» текущей главы:

---

[1] *Jeffrey Dean and Sanjay Ghemawat*. MapReduce: Simplified Data Processing on Large Clusters. Proceedings of the 6th Symposium on Operating Systems Design and Implementation(2004): 137–150.

[2] В этом примере используется версия 0.2.0 библиотеки spark-sklearn.

```
from sklearn.svm import SVC
import numpy as np
from time import time
from spark_sklearn import GridSearchCV # This is the only changed line

# Определение словаря, содержащего все проверяемые значения гиперпараметра
hyperparam_grid = {
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'gamma': np.linspace(0.001, 0.01, num=10),
    'C': np.linspace(1, 10, num=10),
    'tol': np.linspace(0.001, 0.01, 10)
}

classifier = GridSearchCV(svc, hyperparam_grid)

start = time()
classifier.fit(X_train, y_train)
elapsed = time() - start
...
print('elapsed:  %.2f seconds'  % elapsed)

> elapsed: 1759.71 seconds
> Best Kernel: rbf
> Best Gamma: 0.001
> Best C: 2.0
> Accuracy: 0.991
```

В словаре `hyperparam_grid`, передаваемом в `GridSearchCV`, определены значения для четырех гиперпараметров, которые должен обработать алгоритм оптимизации. Всего существует 4000 неповторяющихся сочетаний значений, для полной обработки которых потребуется 1759,71 секунды на одном восьмиядерном компьютере[1] с использованием класса `GridSearchCV` из библиотеки scikit-learn. Если вместо этого класса воспользоваться классом `GridSearchCV` из библиотеки spark-sklearn (как в предыдущем фрагменте кода) и выполнить программу в Spark-кластере из пяти узлов (один главный, четыре рабочих, все узлы того же типа, что и компьютер в предыдущем примере), то наблюдается почти линейное ускорение – задачи выполняются только на четырех рабочих узлах:

```
> elapsed: 470.05 seconds
```

Хотя библиотека spark-sklearn очень удобна в использовании и позволяет распараллеливать процесс оптимизации гиперпараметров в кластере машин (узлов) с минимальными затратами на разработку, набор предлагаемых ею функциональных возможностей относительно невелик[2]. Более того, эта библиотека предназначена для обработки наборов данных, полностью умещающихся в оперативной памяти, что ограничивает область ее применения. Для более крупномасштабных промышленных приложений Spark ML предлагает более солидный

---

[1] Восемь процессоров Intel Broadwell CPU, 30 Гб оперативной памяти.

[2] Отметим, что в библиотеке spark-sklearn не реализованы отдельные алгоритмы обучения, такие как методы опорных векторов или метод k-средних. В настоящее время реализованы только простые и легко распараллеливаемые задачи, подобные кросс-валидации поиска по решетке.

набор алгоритмов параллелизации, реализованных и оптимизированных для запуска в форме заданий в MapReduce-стиле в распределенных кластерах Spark. Как одна из наиболее тщательно проработанных и широко известных распределенных программных сред машинного обучения, Spark ML не только предоставляет общеизвестные алгоритмы машинного обучения для классификации и кластеризации, но также поддерживает функциональные возможности для распределенного извлечения и преобразования данных, позволяя создавать конвейеры для гибкой и удобной в сопровождении обработки, а также дает возможность сохранять сериализованные версии объектов машинного обучения для выполнения контрольных проверок и перемещения на другие платформы.

Попробуем применить некоторые API Spark ML к тому же набору данных для классификатора спама, который использовался в главе 1 и в предыдущем разделе «Генерация объяснений с помощью LIME» текущей главы. Основное внимание будет сосредоточено на использовании конвейеров Spark ML для формирования непрерывного потока разработки. Подобно конвейерам scikit-learn (**https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html**), конвейеры Spark ML позволяют объединить несколько последовательных операций в единый логически согласованный поток, поддерживаемый универсальным API-интерфейсом. Конвейеры работают в специальных фреймах Spark DataFrame (**https://spark.apache.org/docs/latest/sql-programming-guide.html**), представляющих собой оптимизированные наборы данных, организованных в виде столбцов, похожие на фреймы Pandas DataFrame, но с поддержкой преобразований Spark. Ниже приведена реализация конвейера разработки классификатора спама с использованием Spark ML, но пропущен этап синтаксического анализа сообщений электронной почты и форматирования кода набора данных, поскольку используется код из предыдущих примеров[1]:

```
from pyspark.sql.types import *
from pyspark.ml import Pipeline
from pyspark.ml.feature import Tokenizer, CountVectorizer
from pyspark.ml.classification import RandomForestClassifier
from pyspark.ml.evaluation import BinaryClassificationEvaluator

# Считывание исходных необработанных данных
X, y = read_e-mail_files()

# Определение схемы DataFrame для присваивания имен и типов каждому столбцу
# в объекте DataFrame, который будет создан
schema = StructType([
            StructField('id', IntegerType(), nullable=False),
            StructField('e-mail', StringType(), nullable=False),
            StructField('label', DoubleType(), nullable=False)])

# Создание представления данных Spark DataFrame с тремя столбцами, индексом,
# текстом сообщения e-mail и числовой меткой
df = spark.createDataFrame(zip(range(len(y)), X, y), schema)
```

[1] Полный код примера можно найти в виде Python Jupyter notebook chapter7/spark-mllib-spam-fighting.ipynb (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter7/spark-mllib-spam-fighting.ipynb**) в репозитории кода для данной книги.

```
# проверка созданной схемы, чтобы убедиться в том, что все выполнено правильно
df.printSchema()
> root
  |-- id: integer (nullable = false)
  |-- e-mail: string (nullable = false)
  |-- label: double (nullable = false)
```

Небольшое замечание: Spark ML требует, чтобы метки имели тип Double. (Если неправильно определить тип меток, то при выполнении конвейера возникают ошибки.) В предыдущем примере мы создали список `StructType`, который передается как схема в функцию `spark.createDataFrame()` для преобразования набора данных в формате списка Python объекта Spark `DataFrame`. После преобразования данных в формат, понятный Spark, можно определить конвейер (почти все классы Spark ML поддерживают функцию `explainParams()` или `explainParam(paramName)`, которая в удобном виде выводит соответствующие фрагменты документации с описаниями параметров для этого класса – очень полезная возможность, особенно с учетом того, что подробную документацию на Spark ML иногда трудно найти):

```
# Разделение случайным образом набора данных на тренировочный и тестовый наборы
# с коэффициентом TRAINING_SET_RATIO=0.7 (seed устанавливается для воспроизводимости)
train, test = df.randomSplit([TRAINING_SET_RATIO, 1-TRAINING_SET_RATIO], seed=123)

# Во-первых, преобразование (токенизация) строки e-mail (преобразование букв
# в нижний регистр, затем разделение по пробельным символам)
tokenizer = Tokenizer()

# Во-вторых, преобразование токенов в count-векторы
vectorizer = CountVectorizer()

# В-третьих, применение функции оценки RandomForestClassifier
rfc = RandomForestClassifier()

# Наконец, создание конвейера
pipeline = Pipeline(stages=[tokenizer, vectorizer, rfc])
```

Удобной функциональной возможностью конвейеров Spark ML является возможность определения параметров для компонентов конвейера в словаре параметров, который может быть передан в конвейер при выполнении, что позволяет четко разделить логику приложения и настраиваемые параметры. Это может показаться не слишком значимой функциональной возможностью, но может сформировать совершенно другой подход к сопровождению кода. Отметим, что в приведенном примере не определялись никакие параметры при инициализации компонентов конвейера (`Tokenizer`, `CountVectorizer`, `RandomForestClassifier`) – даже если бы мы определили какие-то параметры, они были бы замещены (перезаписаны) значениями параметров, переданными при вызове функции `pipeline.fit()`, которая непосредственно выполняет конвейер:

```
# Определение словаря для передачи значений параметров компонентов конвейера
paramMap = {
    tokenizer.inputCol: 'e-mail',
    tokenizer.outputCol: 'tokens',

    vectorizer.inputCol: 'tokens',
    vectorizer.outputCol: 'vectors',

    rfc.featuresCol: 'vectors',
```

```
    rfc.labelCol: 'label',
    rfc.numTrees: 500
}

# Применение всех параметров в конвейере, выполнение конвейера и подгонка модели
model = pipeline.fit(train, params=paramMap)
```

Теперь у нас есть тренированная модель конвейера, которую можно использовать для выполнения прогнозирования. Запустим пакетное прогнозирование на нашем тестовом наборе данных и вычислим оценку результата, используя объект `BinaryClassificationEvaluator`, который автоматизирует все преобразования данных, необходимые для генерации числовых оценочных метрик:

```
# Выполнение прогнозирования на тестовом наборе данных
prediction = model.transform(test)

# Оценка результатов с использованием удобного объекта Evaluator
evaluator = BinaryClassificationEvaluator(rawPredictionCol='rawPrediction')
pr_score = evaluator.evaluate(prediction,
{evaluator.metricName: 'areaUnderPR'})
roc_score = evaluator.evaluate(prediction,
{evaluator.metricName: 'areaUnderROC'})

print('Area under ROC curve score: {:.3f}'.format(roc_score))
print('Area under precision/recall curve score: {:.3f}'.format(pr_score))

> Area under ROC curve score: 0.971
> Area under precision/recall curve score: 0.958
```

С помощью Spark ML мы написали компактный, но легко масштабируемый фрагмент кода, который способен обрабатывать чрезвычайно большие объемы данных[1]. Конвейер Spark ML помогает создать простую и понятную структуру кода, которая будет весьма полезной при расширении кодовой базы. Можно также добавить в конвейер логику оптимизации гиперпараметров посредством конфигурирования объекта `ParamGridBuilder` (для определения значений-кандидатов для гиперпараметров) и объекта `CrossValidator` или `TrainValidationSplit` (для вычисления эффективности гиперпараметров/механизма оценки)[2].

Платформа Spark предоставляет удобные способы применения распараллеливания и кластерных вычислений для сокращения времени задержек и улучшения масштабирования систем машинного обучения. Программирование распределенных приложений значительно сложнее локальной разработки с использованием библиотеки scikit-learn, но затраты и усилия по его освоению непременно многократно окупятся в будущем.

Использование облачных сервисов

Рынку услуг «машинное обучение как сервис» предсказывают рост до 20 миллиардов долларов к 2025 году. Все известные провайдеры крупных облачных сервисов

---

[1] Выполнение этого примера было проверено в кластере Spark из пяти узлов (один главный, четыре рабочих) с использованием механизма DataProc корпорации Google.

[2] Подробное описание всех этих объектов см. в соответствующей документации (**https://spark.apache.org/docs/2.2.0/api/java/org/apache/spark/ml/tuning/package-summary.html**).

предлагают некоторый набор услуг в области машинного обучения и инфраструктуры данных, которые вы можете использовать для быстрого и экономичного масштабирования своих операций. Такие сервисы освобождают организации от операционных издержек по сопровождению и управлению кластером Spark или по развертыванию среды TensorFlow, которые требуют значительных трудозатрат на конфигурирование и обслуживание.

Главные действующие лица в сфере общедоступных облачных сервисов, такие как Amazon Web Services (AWS) и Google Cloud Platform (GCP), предоставляют мощные API для анализа видео, речи и изображений с использованием предварительно тренированных моделей машинного обучения. Кроме того, они предлагают бессерверные интерфейсы для выполнения экспериментальных или производственных заданий с применением методов машинного обучения без необходимости установления канала связи по протоколу Secure Shell (SSH) с экземпляром объекта для установки зависимостей или процессов перезагрузки. Например, Google Cloud Dataflow (**https://cloud.google.com/dataflow/**) – это полностью управляемая платформа, позволяющая пользователям выполнять задания, написанные в универсальной модели программирования Apache Beam (**https://beam.apache.org/**), не беспокоясь о загрузке и обеспечении производительности. Имея возможность десятикратного масштабирования пропускной способности, мы просто должны изменить один параметр для запуска приблизительно в 10 раз большего количества экземпляров для регулирования нагрузки. Google Cloud Dataproc (**https://cloud.google.com/dataproc/**) – сервис, управляемый Spark или Hadoop, который позволяет пользователю развертывать крупные кластеры машин (предварительно загруженные и сконфигурированные с помощью Spark, Hadoop, Pig, Hive, Yarn или другими инструментальными средствами поддержки распределенных вычислений) «в среднем менее чем за 90 секунд». Например, настройка Spark-кластера из пяти узлов на Dataproc для выполнения примера классификации спама с использованием Spark ML из предыдущего раздела заняла менее минуты после выполнения в командной строке следующей команды:

```
gcloud dataproc clusters create cluster-01 \
    --metadata "JUPYTER_CONDA_PACKAGES=numpy:pandas:scipy:scikit-learn" \
    --initialization-actions \
        gs://dataproc-initialization-actions/jupyter/jupyter.sh \
    --zone us-central1-a \
    --num-workers 4 \
    --worker-machine-type=n1-standard-8 \
    --master-machine-type=n1-standard-8
```

Эта команда создания кластера позволяет пользователям определить действия при инициализации `initialization-actions` – скрипт для установки специальных пакетов и зависимостей для данных/кода, который будет выполнен на этапе подготовки всего необходимого обеспечения на каждой машине в кластере. В приведенном выше примере команды использовался скрипт `initialization-actions` для установки Jupyter notebook и пакетов языка Python Pandas, SciPy и т. п. по зависимостям.

Amazon Machine Learning (**https://aws.amazon.com/machine-learning/**) позволяет даже новичкам воспользоваться преимуществами машинного обучения, выгру-

жая данные на необходимые им платформы (например, S3 или Redshift) и «создавая» модель машинного обучения посредством настройки нескольких предварительных параметров с помощью веб-интерфейса. Google Cloud ML Engine (**https://cloud.google.com/ml-engine/**) обеспечивает гораздо бóльшую гибкость, предоставляя пользователям возможность запускать собственный код тренировки модели TensorFlow на бессерверной архитектуре, затем сохранять тренированную модель и экспонировать ее через API-прогнозирования. Благодаря наличию такой инфраструктуры инженеры по машинному обучению могут сосредоточиться исключительно на обеспечении эффективности своих алгоритмов и воспользоваться предоставляемыми операционными функциями развертывания и масштабирования системы машинного обучения.

Использование облачных сервисов в организациях может обеспечить гораздо бóльшую гибкость при экспериментах с решениями задач машинного обучения. Часто эти экспериментальные решения становятся даже более экономичными, после того как вы рассмотрите все операционные и эксплуатационные издержки, присущие ручному управлению развертыванием систем машинного обучения. Для организаций, которые вынуждены работать с различными вариантами реализации и архитектур систем машинного обучения или с системами, для которых необходимо значительное масштабирование в течение короткого интервала времени, использование общедоступных облачных сервисов, таких как Google Cloud ML Engine, имеет немалое значение. Но доступность таких сервисов полностью зависит от деловых потребностей и целей организаций, которые их поддерживают и финансируют (т. е. насколько прибыльны эти сервисы для Amazon, Google, Microsoft и т. д.), поэтому создание важных служб обеспечения безопасности на основе этих сервисов, возможно, является не самым оптимальным стратегическим решением для любой организации.

Удобство сопровождения

Успешно эксплуатируемые системы машинного обучения часто переживают своих создателей (в пределах организации). В подобных случаях такие системы должны обслуживаться инженерами, для которых нет необходимости понимать, почему при разработке были выбраны те или иные решения. Удобство сопровождения является одним из принципов создания программного обеспечения, который расширяет границы сфер обеспечения безопасности и машинного обучения. Все программные системы должны быть оптимизированы для удобства сопровождения, поскольку системы с неудачно организованным сопровождением в конечном итоге деградируют и завершают свой жизненный цикл. Гораздо хуже то, что такие системы могут влачить свое жалкое существование в течение десятилетий, истощая ресурсы организаций и препятствуя достижению поставленных целей. Недавно опубликованный корпорацией Google материал[1] подтверждает, что из-за сложности и зависимости от постоянно изменяющихся данных системы машинного обучения даже более чувствительны, чем прочие системы, к накоплению технического долга.

---

[1] *D. Sculley et al*. Hidden Technical Debt in Machine Learning Systems. Proceedings of the 28th International Conference on Neural Information Processing Systems (2015): 2503–2511.

В этом разделе кратко рассматриваются некоторые концепции удобства сопровождения. Мы не углубляемся в подробности, так как большинство этих концепций подробно описано в специализированных публикациях по этой теме[1].

Проблема: проверка контрольных точек, управление версиями и развертывание моделей

Модель машинного обучения – это код или данные? Поскольку модели тесно связаны с природой данных, используемых для их генерации, этот факт позволяет обосновать мнение, что модели должны восприниматься как данные, так как код должен быть независимым от обрабатываемых данных. Но в рассматриваемых моделях существует значимая операционная связь с теми же процессами управления версиями и развертывания, которые выполняются для соответствующего исходного кода. Наше мнение таково: модели машинного обучения должны рассматриваться как совокупность кода и данных. Хранение параметров и гиперпараметров модели в системах управления версиями, таких как Git, делает весьма удобным восстановление предыдущих версий моделей, если что-то пошло не так. Хранение моделей в базах данных позволяет выполнять запросы параметров для различных версий в параллельном режиме, что может становиться чрезвычайно ценной возможностью в некоторых ситуациях.

Для целей аудита и разработки полезно убедиться в том, что каждое решение, принимаемое системой в любой момент времени, может быть повторно воспроизведено. Например, рассмотрим сервер выявления аномалий в веб-приложениях, который помечает конкретный сеанс пользователя как аномальный. Из-за сильной неустойчивости входных данных, которую можно наблюдать в веб-приложениях, эта система пытается непрерывно оценивать (измерять) меняющийся трафик и адаптироваться к нему с помощью непрерывной автоматической точной настройки параметров. Более того, модели машинного обучения постоянно регулируются и усовершенствуются со временем инженерами-людьми или с помощью механизмов автоматизированного обучения. Проверка контрольных точек и управление версиями моделей позволяет увидеть, что этот пользовательский сеанс был зафиксирован моделью два месяца назад.

Сериализация моделей для хранения может быть такой же простой, как использование интерфейса сериализации объектов `pickle` в языке Python (**https://docs.python.org/2/library/pickle.html**). Для эффективного использования пространства хранения и быстрого восстановления, а также для улучшения переносимости можно воспользоваться специализированным форматом хранения, который сохраняет всю информацию о параметрах, требуемую для восстановления модели машинного обучения. Например, хранение всех весовых коэффициентов признаков тренированной модели линейной регрессии в файле формата JSON является не зависимым от платформы и программной среды способом сохранения и восстановления линейных регрессоров.

Predictive Model Markup Language (PMML) – ведущий открытый стандарт сериализации на основе XML и совместного использования моделей прогнозируе-

[1] *Joost Visser et al.* Building Maintainable Sotware, Java Edition: Ten Guidelines for Future-Proof Code (Sebastopol, CA: O'Reilly Media, 2016).

мого майнинга данных[1]. Помимо хранения параметров модели, этот формат также способен кодировать разнообразные преобразования, применяемые к данным на этапах предварительной обработки и обработки итоговых результатов. Полезной функциональной характеристикой формата PMML является возможность разработки модели с использованием одной инструментальной программной среды машинного обучения и развертывание готовой системы в другой программной среде поддержки машинного обучения. В качестве обобщающего звена для различных систем PMML позволяет разработчикам сравнивать эффективность и точность одной и той же модели, работающей в различных программных средах поддержки машинного обучения.

Механизм развертывания моделей машинного обучения должен быть оснащен так называемой «защитой от дурака» во всех случаях, когда это возможно. Системы машинного обучения могут развертываться как веб-сервисы (например, доступные через REST API) или как встроенные компоненты ПО внутреннего звена. Тесная связь с другими системами является отрицательным фактором, поскольку приводит к многочисленным нестыковкам при развертывании, что дает потерю гибкости всей программной системы в целом. Обеспечение доступа к системам машинного обучения через API добавляет весьма полезный уровень косвенности, способный придать бóльшую гибкость при развертывании, A/B-тестировании и отладке.

Цель: амортизация отказов

Программные системы должны обрабатывать критические отказы и сбои аккуратно и прозрачно. Если более продвинутая и востребованная версия веб-сайта не работает в старом браузере, то вместо нее должна быть предоставлена более приемлемая упрощенная версия. Системы машинного обучения не являются исключением. Амортизация отказов в работе является важной функциональной характеристикой систем, представляющих ключевое звено, которое может стать потенциальной причиной отказа в доступе других систем. Системы обеспечения безопасности часто расположены в критических точках, следовательно, должны иметь четко определенную стратегию обработки отказов и сбоев.

Должны ли системы обеспечения безопасности работать в режиме отказоустойчивости (пропускать запросы, если система отказывает в ответе) или отказобезопасности (блокировать все запросы, если система отказывает в ответе)? На этот вопрос невозможно ответить без тщательного и полного исследования приложения, взвешенной оценки риска и размера издержек при отражении атаки по сравнению с издержками, связанными с отказом в доступе к приложению реальным пользователям. Например, система аутентификации, вероятнее всего, будет работать в режиме отказобезопасности, так как режим отказоустойчивости позволит всем получать доступ к защищаемым ресурсам. Но система выявления спама в электронной почте должна работать в режиме отказоустойчивости, поскольку блокирование всего сервиса электронной почты приведет к гораздо более крупным издержкам, чем пропуск некоторой части спама. В общем случае цена компрометации (взлома) системы значительно превышает цену отказа в обслу-

[1] *Alex Guazzelli et al*. PMML: An Open Standard for Sharing Models. The R Journal 1 (2009): 60–65.

живании пользователей, поэтому в системах обеспечения безопасности предпочтение обычно отдается стратегии отказобезопасности. Но в некоторых ситуациях такой подход может сделать систему уязвимой к атакам типа DoS, потому что атакующие просто выводят из строя шлюз защиты, перекрывая добропорядочным пользователям доступ ко всей системе в целом.

Амортизация отказов систем обеспечения безопасности также может быть обеспечена наличием заменяющих упрощенных резервных систем. Например, рассмотрим случай, когда на вашем сайте наблюдаются значительные объемы трафика, а система машинного обучения, которая отличает трафик, генерируемый пользователями-людьми, от трафика ботов, подвержена риску потери устойчивости при высоком уровне нагрузки. Возможно, разумным решением является возврат к более простой (и даже примитивной) и менее ресурсозатратной стратегии CAPTCHA до тех пор, пока трафик не вернется в нормальное русло.

Тщательно продуманная стратегия обеспечения непрерывной защиты системы при отказах компонентов подсистемы безопасности чрезвычайно важна, поскольку любые лазейки в комплексе обеспечения безопасности (например, сниженная доступность системы) предоставляют возможности их использования атакующими.

Цель: легкость настройки и конфигурации

Строгое разделение кода и конфигурации является основным требованием для всех программ промышленного (корпоративного) уровня. Этот принцип особенно важен для систем машинного обучения, предназначенных для обеспечения безопасности. В сфере защищенных операций конфигурации систем обеспечения безопасности должны точно настраиваться аналитиками безопасности операций, которые не обязаны знать тонкости разработки ПО. Правильное проектирование ПО и конфигурации, которое позволяет аналитикам точно настраивать системы без привлечения инженеров ПО, может существенно сократить операционные издержки для таких систем и обеспечить более гибкую и универсальную организацию.

Мониторинг и система оповещения

Системы машинного обучения для обеспечения безопасности должны быть быстрыми и надежными. В идеальном случае в таких системах не должны допускаться какие-либо простои, а прогнозы должны выполняться практически в реальном времени[1]. Но случайные неудачи, которые приводят к снижению эффективности или полному отказу системы, неизбежны. Способность своевременно выявлять подобные события позволяет обеспечить меры, ограничивающие отрицательное воздействие, например создание резервных систем и привлечение операционного персонала к расследованию причин возникшей проблемы.

---

[1] «Architecting a Machine Learning System for Risk» (**https://medium.com/airbnb-engineering/architecting-a-machine-learning-system-for-risk-941abbba5a60**), авторы Назим Хаким (Naseem Hakim) и Аарон Кейс (Aaron Keys) – подробный обзор, в котором показано, как крупная компания, такая как Airbnb, проектирует работающую в реальном времени систему машинного обучения для обеспечения безопасности и программную среду оценки риска.

Рабочая среда мониторинга (monitoring framework) – это система, которая объединяет метрики из различных источников в одной центральной локации для ручного мониторинга и выполнения выявления аномалий. Такие системы чаще всего состоят из пяти следующих обязательных компонентов:

- сборщики метрик;
- база данных временны́х рядов;
- механизм выявления;
- уровень визуализации;
- механизм оповещения.

Обычный рабочий поток для мониторинга приложения начинается, когда приложения периодически публикуют метрики в специальном пункте сбора рабочей среды мониторинга (например, в конечной точке REST) или когда агенты-сборщики метрик в конечных точках извлекают метрики из системы. Затем эти метрики сохраняются в базе данных временны́х рядов, к которой может обращаться с запросами механизм выявления для генерации предупреждающих оповещений, а уровень визуализации использует базу данных для создания графиков и схем. Кроме того, механизм оповещения отвечает за информирование соответствующих заинтересованных лиц (стейкхолдеров) о наиболее значимых событиях, автоматически выявленных рабочей средой мониторинга.

Рабочие среды мониторинга и оповещения часто вынуждены обеспечивать возможность оповещения администраторов об отказах других систем, но не способны предупреждать о собственных возникающих отказах. Хотя невозможно полностью устранить этот риск, важно проектировать или выбирать системы мониторинга, которые сами по себе обладают высокой доступностью, надежностью и масштабируемостью. Добавление избыточности в решения по мониторингу также может снизить вероятность полной потери видимости при отказе единственной машины. Подробное обсуждение проблем мониторинга не относится к теме данной книги, но требует времени на исследование и усилий для более глубокого изучения эффективного мониторинга и механизма оповещения[1]. На начальном этапе освоения можно порекомендовать наиболее известные рабочие среды мониторинга, такие как Prometheus (**https://prometheus.io/**), стек TICK (**https://www.influxdata.com/**), Graphite (**https://graphiteapp.org/**) и Grafana (**https://grafana.com/**).

Эффективность и доступность являются не единственными свойствами системы, которые требуют мониторинга. Поскольку статистические системы принимают на обработку данные из реального окружающего мира, которым свойственна определенная степень непредсказуемости, также важен мониторинг общей эффективности (производительности) системы в целом для гарантии того, что система постоянно выдает надежные и актуальные результаты. Эта задача редко является вполне определенной и понятной, так как измерения эффективности (производительности) непременно требуют особого доступа к системе для надежной проверки корректности ее прогнозов и частого использования меток, присвоенных человеком, из циклов обратной связи. Общепринятым приближенным методом измерений меняющейся производительности и эффективности является мониторинг распределения прогнозов, выполняемых системой. Например,

[1] *Slawek Ligus*. Effective Monitoring and Alerting for Web Operations (Sebastopol, CA: O'Reilly Media, 2012).

если в системе обычно наблюдается 0,1 % запросов на вход (регистрацию), помеченных как подозрительные, то неожиданный скачок этого значения до 5 %, вероятнее всего, заслуживает более тщательного исследования[1].

Другая важная функция – возможность мониторинга изменений во входных данных независимо от выходных данных, генерируемых системой машинного обучения. Свойства данных, такие как статистическое распределение, объем, скорость поступления и плотность, могут оказывать существенное воздействие на эффективность и производительность систем машинного обучения. Изменения в распределении данных по времени могут повлиять на сдвиг трендов, получение новых источников данных (например, новый клиент-покупатель или приложение, передающие данные в систему) или в редких случаях приводить к преднамеренному загрязнению данных (атаки типа red herring, которые рассматриваются в главе 8). Также часто встречается увеличение плотности (или, наоборот, разреженности) входных данных, которое оказывает отрицательное воздействие на системы машинного обучения.

Конвейеры сбора данных и извлечения признаков теряют работоспособность, если не поддерживается их адаптация к изменениям форматов данных. Например, механизм извлечения признаков веб-приложения, собирающий IP-адреса из запросов HTTP, может предполагать, что все IP-адреса существуют только в формате IPv4. Когда веб-сайт начинает поддерживать формат IPv6, это предположение становится неверным, и наблюдается массовое увеличение точек данных с незаполненным (нулевым) полем IP-адреса. Несмотря на возможные трудности учета и поддержки изменений форматов входных данных, мониторинг возникновения пропущенных значений полей в наборе извлекаемых признаков является эффективным промежуточным средством. Изменение тренда в счетчиках ошибок или исключений в отдельных компонентах системы (например, в конвейере извлечения признаков) также может стать успешным средством раннего выявления критических сбоев системы, поэтому такие счетчики должны быть стандартными отслеживаемыми метриками в правильно организованных производственных системах.

Безопасность и надежность

Где бы ни развертывались решения по обеспечению безопасности, всегда следует ожидать возникновения вредоносной деятельности. Рассмотрим гарантии безопасности и секретности, которые могут обеспечить специализированные системы машинного обучения.

Функция: устойчивость и надежность работы во враждебных средах

Системы обеспечения безопасности подвержены постоянному риску враждебного воздействия. Атакующие всегда мотивированы и стремятся преодолеть за-

[1] Резкий скачок или аномалия такого типа означает: что-то изменилось в данных или в системе. Это могло произойти из-за атаки на конечную точку входа (регистрации) на сайте или из-за трудно обнаруживаемых проблем, например из-за некорректно тренированной или настроенной модели, результатом чего стал резкий рост ложноположительных результатов на фоне ранее наблюдавшейся картины.

щитные барьеры, поскольку по своей сущности их деятельность направлена на получение вероятной выгоды на атакуемой стороне. Следовательно, необходимо обеспечить надежность и устойчивость производственных систем и их защиту от вредоносной деятельности, пытающейся нарушить производительность, доступность и эффективность таких систем.

Очень важно особо выделить разрушающие воздействия, которые активные противники могут оказать на модели машинного обучения. Это обширное поле исследований в рассматриваемой здесь области, позволяющее понять, какой ущерб могут нанести злоумышленники даже при минимальном доступе к ресурсам и минимальной информации. В частности, для систем машинного обучения для обеспечения безопасности важно заранее понять логику и возможности атакующих. Поэтому вы должны с особой тщательностью выбирать надежные и устойчивые алгоритмы и проектировать системы с надлежащими проверками и балансами, позволяющими выявлять попытки мошенничества и ограничивать их воздействия.

Огромное разнообразие статистических атак оказывает отрицательное воздействие на системы машинного обучения, снижая их стабильность и надежность. Проектировщики и разработчики систем машинного обучения для обеспечения безопасности занимают особое, отличающееся от прочих положение, поскольку им приходится защищать свои системы от враждебного воздействия. Более подробно состязательное машинное обучение во враждебных средах рассматривается в главе 8, но здесь необходимо отметить важность обеспечения устойчивости и надежности систем машинного обучения для обеспечения безопасности к атакам различных типов.

Функция: защита и гарантии секретности данных

Секретность данных – особая область науки о данных, важность которой увеличивается по мере того, как технологии становятся все более распространенными и захватывают все сферы жизни. Системы машинного обучения обычно находятся в противоречии с концепцией обеспечения секретности, поскольку алгоритмы успешнее работают с более подробными данными. Например, возможность доступа к обширным аудио- и видеофрагментам, отснятым на мобильных устройствах, может предоставить обширнейший исходный материал для классификации законности запросов к API мобильных приложений, выполняемых в конечной точке входа в учетную запись, но настолько открытый доступ обычно считается грубым нарушением приватности, поэтому редко осуществляется на практике.

В дополнение к проблемам сохранения секретности, связанным со сбором приватных данных от пользователей и в конечных точках, существует также проблема утечки информации из самих тренируемых моделей машинного обучения[1]. Некоторые модели машинного обучения генерируют выходные данные, позволяющие внешнему наблюдателю без затруднений вывести или восстановить исходный набор тренировочных данных или тестовых данных, на основе которых был сгенерирован итоговый прогноз. Например, алгоритм k-ближайших соседей (k-NN) и метод опорных векторов на основе ядра особенно уязвимы к утечкам информации, потому что некоторые тренировочные дан-

[1] *Daniel Hsu*. Machine Learning and Privacy. Columbia University, Department of Computer Science.

ные могут быть выведены из методов и функций вычисления плотности, которые представляют опорные векторы[1].

Задача создания алгоритмов машинного обучения с обеспечением секретности уже стала областью активных исследований, но для нее трудно найти решение, поскольку атакующие часто получают доступ к общей (глобальной) информации. Если атакующий получил доступ к тренируемой модели машинного обучения и к 50 % (и больше) тренировочных данных, у него появляется возможность восстановления остальных 50 % данных с высокой степенью достоверности. Дифференциальная приватность (differential privacy)[2] обозначает класс решений задачи машинного обучения с обеспечением секретности, который направлен на устранение этой проблемы посредством создания трудностей для атакующего, т. е. снижения достоверности восстановления недостающих элементов информации по доступным элементам данных.

Обеспечение секретности в системах машинного обучения должно входить в список важнейших требований, поскольку нарушение приватности и уязвимость секретных данных обычно приводят к серьезным и дорогостоящим последствиям. Производственные системы должны обеспечивать защитные средства и гарантии сохранения секретности, основанные на надежных и проверенных теоретических и технических рабочих программных средах, и ограничивать ущерб, который могут причинить атакующие, похитив секретную информацию.

Обратная связь и удобство использования

Удобство работы пользователя с особым выделением обмена информацией и взаимодействия человека и компьютера при сбалансированной автоматизации и деятельности (восприятия) пользователя являются истинными признаками успешной системы машинного обучения для обеспечения безопасности. Между людьми и компьютерами (и машинами вообще) существуют неизбежные противоречия. Решения на основе машинного обучения не смогут реализовать весь свой потенциал, если не обеспечат удобство работы пользователя с такой постоянно развивающейся системой. Объяснимость результатов – важное предварительное требование для обеспечения доверия, потому что большинство пользователей не будет доверять результатам, полученным от систем машинного обучения, если не понимают, как система пришла к такому результату. Прозрачность – это ключ к использованию полной мощности, которую способны предоставить системы машинного обучения. Если система выявления мошеннических попыток входа (регистрации) использует машинное обучение для определения степени подозрительности конкретных попыток входа, то она должна по возможности объяснять пользователю причины принятия своего решения и предлагать меры по устранению возникшей проблемы.

Разумеется, полная объяснимость противоречит принципам обеспечения безопасности, следовательно, системы должны давать минимум информации

---

[1] *Zhanglong Ji, Zachary C. Lipton and Charles Elkan*. Differential Privacy and Machine Learning: A Survey and Review (2014).

[2] *Cynthia Dwork and Aaron Roth*. The Algorithmic Foundations of Differential Privacy. Foundations and Trends in heoretical Computer Science 9 (2014): 211–407.

предполагаемым атакующим нарушителям. Захват атакующими канала обратной связи позволяет им выполнять быстрые итерации и воспользоваться уязвимостями, чтобы в конечном итоге обмануть систему. Возможным решением является регулирование степени прозрачности решений механизма машинного обучения в обратной зависимости от того, насколько вероятным является ее использование атакующей стороной. Если система способна классифицировать типичное поведение атакующего с высокой степенью достоверности, а большинство ложноположительных результатов не являются «положительными с высокой достоверностью», то можно применить стратегию дискриминации прозрачности, которая не позволит явным атакующим злоумышленникам захватить какой-либо канал обратной связи. Такой подход обеспечивает определенную гибкость в снижении отрицательных воздействий неправильных прогнозов, сделанных системами машинного обучения.

Представление информации при взаимодействии человека и компьютера в системах машинного обучения является областью исследований, которой часто пренебрегают. Неправильное отношение к предубеждениям, доверию и динамическому распределению обязанностей между людьми и системами машинного обучения может привести к полной деградации последних.

Резюме

Системы машинного обучения для обеспечения безопасности должны стать одной из прочнейших обязательных связей в современной среде прикладного ПО. Следовательно, такие системы должны соответствовать стандартам качества, масштабируемости и удобства сопровождения, которые превосходят большинство других эксплуатируемых компонентов. В этой главе рассматривалась рабочая среда для исследования и оценки готовности систем к промышленной эксплуатации. Теперь ваша задача как ученых и инженеров в области обеспечения защиты данных – обеспечить полную готовность к реальной эксплуатации развертываемого ПО.

Глава 8

Состязательное машинное обучение

В настоящее время методы машинного обучения начинают повсеместно применяться в особо важных системах, поэтому надежность и стабильность систем машинного обучения заслуживает тщательного исследования. При этом важно не впадать в панику, хотя угроза вмешательства враждебных сил в работу систем машинного обучения вполне реальна. Подобно тому, как взломщик использует уязвимости сетевого защитного экрана, чтобы получить доступ к веб-серверу, сами по себе системы машинного обучения могут стать объектами атаки злоумышленников, преследующих собственные корыстные цели. Таким образом, прежде чем выводить эти решения на передовую линию защиты, чрезвычайно важно подробно исследовать все их слабости и понять, насколько уязвимыми они могут стать при атаках.

Состязательное машинное обучение (adversarial machine learning) – это исследование уязвимостей систем машинного обучения во враждебных (сопернических) средах. Исследователи проблем безопасности и машинного обучения опубликовали множество результатов исследований по реально совершенным атакам против антивирусных механизмов машинного обучения[1], фильтров спама[2], детекторов вторжения в сети, классификаторов изображений[3], анализаторов общественного мнения и настроения[4] и т. п. В последнее время это стало активно развивающейся областью исследований, даже несмотря на то что на практике подобные атаки наблюдаются редко. Когда опасности подвергается приватность информации, национальные суверенитеты и жизни людей, проектировщики и разработчики систем машинного обучения несут ответственность за предотвращение атак и создание защитных средств для этих систем.

---

1 *Weilin Xu, Yanjun Qi and David Evans*. Automatically Evading Classifiers: A Case Study on PDF Malware Classifiers. Network and Distributed Systems Symposium 2016, 21–24 February 2016, San Diego, California.

2 *Blaine Nelson et al*. Exploiting Machine Learning to Subvert Your Spam Filter. Proceedings of the 1st USENIX Workshop on Large-Scale Exploits and Emergent hreats (2008): 1–9.

3 *Alexey Kurakin, Ian Goodfellow and Samy Bengio*. Adversarial Examples in the Physical World (2016).

4 *Bin Liang et al*. Deep Text Classification Can Be Fooled (2017).
Hossein Hosseini et al. Deceiving Google's Perspective API Built for Detecting Toxic Comments (2017).

Уязвимости в системах машинного обучения могут возникать из-за ошибок при проектировании, из-за неизбежных алгоритмических ограничений или из-за сочетания обоих этих факторов. В этой главе рассматриваются некоторые внутренние уязвимости и атаки на алгоритмы машинного обучения. Затем мы воспользуемся прибретенными знаниями для проектирования систем, более устойчивых к атакам.

Терминология

Предшествующие исследования в области состязательного машинного обучения позволили определить систематическую классификацию качественно анализируемых атак на системы машинного обучения на основе трех измерений свойств[1]:

- воздействие (influence):
  - каузативные атаки (causative attacks) обозначают попытки враждебной стороны воздействовать на процесс тренировки посредством искажения тренировочных данных или параметров тренировочного этапа. Так как противнику трудно совершать какие-либо действия с набором тренировочных данных, управляемых в офлайновом режиме, этот тип атак преимущественно направлен на онлайновые механизмы обучения. Онлайновые механизмы обучения автоматически адаптируются к изменяющимся распределениям данных, напрямую используя взаимодействие с пользователем или обратную связь при прогнозировании для обновления тренируемой модели. Жертвуя стабильностью ради адаптируемости, такие системы обучения непрерывно развиваются при помощи постепенной тренировки статистических моделей на постоянно обновляющихся актуальных данных. Типичным практическим примером онлайнового обучения является сервис классификации изображений, который обучается по корректировкам и подкреплениям пользователей, или система выявления вредоносного трафика на веб-сайтах с часто наблюдаемыми пиковыми периодами вирусного (или мошеннического) трафика;
  - пробные, или разведочные, атаки (exploratory attacks) основаны исключительно на взаимодействиях с системами машинного обучения на послетренировочном этапе. При этом типе атак противники не оказывают какого-либо воздействия на тренировочный набор данных, а вместо этого находят и используют опасное пространство, чтобы вынуждать модели совершать ошибки, которые они не должны допускать. Простейшим примером пробной атаки является применение фаззинга грубой силы к пространству входных данных классификатора на основе машинного обучения для поиска неправильно классифицированных экземпляров;
- специфика (specificity):
  - целенаправленные атаки (targeted attacks) обозначают попытки создать преднамеренный сдвиг (смещение) прогнозов модели в определенном направлении для изменения итоговых результатов к состоянию, необ-

[1] *Marco Barreno et al*. Can Machine Learning Be Secure? Proceedings of the 2006 ACM Symposium on Information, Computer and Communications Security (2006): 16–25.

ходимому атакующей стороне. Например, целенаправленная атака на классификатор семейств вредоносного ПО может привести к ошибочной классификации вредоносных программ семейства A как принадлежащих семейству B;

- беспорядочные атаки (indiscriminate attacks) – это атаки без конкретно определенных целей, при которых атакующим нужны модели для генерации неправильных решений, но абсолютно не имеет значения, какие именно конечные результаты выдаст система. Беспорядочная атака на классификатор вредоносного ПО из предыдущего примера может привести к неправильной классификации вредоносных программ из семейства A как принадлежащих любому другому семейству;

- нарушение защиты (взлом) (security violation):
  - атаки на целостность (integrity attacks) систем машинного обучения воздействуют только на способность детекторов в системе обеспечения безопасности обнаруживать атаки. Таким образом, они снижают относительную долю истинно положительных результатов (т. е. полноту). Успешное проведение такой атаки на защитный экран веб-приложений на основе машинного обучения означает, что противник может успешно атаковать этот защитный экран, специально предназначенный для выявления атак;
  - атаки на доступность (availability attacks), которые обычно являются результатом беспорядочных атак, ухудшают удобство использования системы, снижая долю истинно положительных результатов и увеличивая долю ложноположительных результатов. Когда системы выводятся из строя таким способом, затрудняются надлежащие действия с полученными результатами, следовательно, атака выглядит как снижение доступности системы. Этот тип атак связан только с казуативными атаками, поскольку обычно подразумевают искажение и нарушение функций принятия решений обучаемого (в онлайн-режиме) агента.

Важность состязательного машинного обучения

Машинное обучение быстро становится обязательным инструментальным средством в арсенале каждого специалиста-практика по обеспечению безопасности, но трое из четырех исследователей все еще осознают, что современные решения в области обеспечения безопасности, управляемые искусственным интеллектом, существенно недоработаны и имеют дефекты[1]. В большинстве случаев недоверие к решениям по обеспечению безопасности на основе машинного обучения возникает из-за легкости, с которой злоумышленники могут обойти эти решения. Любопытный парадокс состоит в том, что многие профессионалы в области обеспечения безопасности также предсказывают, что в будущем решения в этой области будут управляться искусственным интеллектом,

[1] *Carbon Black*. Beyond the Hype: Security Experts Weigh in on Artificial Intelligence, Machine Learning and Non-Malware Attacks (2017). (**https://www.carbonblack.com/2017/03/28/beyond-hype-security-experts-weigh-artificial-intelligence-machine-learning-non-malware-attacks/**).

в частности методами машинного обучения. Необходимость устранения этого разрыва между реалиями сегодняшнего дня и ожиданиями в ближайшем будущем объясняет, почему состязательное машинное обучение столь важно для обеспечения безопасности.

Состязательное машинное обучение трудно реализовать, потому что большинство решений машинного обучения ведет себя как черные ящики. Отсутствие прозрачности того, что происходит внутри детекторов и классификаторов, затрудняет для пользователей и специалистов-практиков понимание смысла прогнозов модели. Более того, отсутствие объяснений решений, принятых этими системами, означает, что пользователям очень трудно определить те случаи, когда система находится под воздействием злоумышленников. Пока люди не могут быть вполне уверенными в надежности систем машинного обучения, всегда будет возникать сопротивление внедрению этих систем и принятию их как основных механизмов принятия решений в области обеспечения безопасности.

Опасные уязвимости в алгоритмах машинного обучения

Системы обеспечения безопасности являются естественными целями для вредоносных и мошеннических действий, потому что атакующие злоумышленники, которые успешно обходят их, чаще всего получают прямую выгоду. Системы, усиленные методами машинного обучения, содержат новую поверхность атак, которую противники могут использовать, когда они владеют базовыми знаниями в этой области. Взлом системных сред с использованием ошибок проектирования и реализации не является чем-то новым, но обман статистических моделей – это совершенно другая деятельность. Для понимания уязвимостей в алгоритмах машинного обучения рассмотрим, как рабочая среда, в которой применяются эти методы, влияет на их эффективность. В качестве аналогии возьмем пловца, который учится и отрабатывает стили плавания в бассейнах в течение всей жизни. Вполне вероятно, что он будет очень сильным пловцом в бассейнах, но если по какой-то случайности неожиданно попадет в открытый океан, то, возможно, окажется не подготовленным к борьбе с суровыми и даже жестокими внешними условиями, хотя будет стараться изо всех сил.

Методики машинного обучения обычно разработаны с предварительными предположениями о стабильности данных, независимости признаков и малой степени стохастичности (случайности). Предполагается, что тренировочный и тестовый наборы данных выбраны из совокупностей, распределения которых не изменяются со временем, а выбранные признаки полагаются независимо и равномерно распределенными. В алгоритмах машинного обучения обычно не предусматривается сохранение эффективности во враждебных средах, где все перечисленные предположения становятся неверными. Попытка адаптировать дескриптивную и устойчивую модель к процессу выявления противников, постоянно изменяющих свое поведение с целью избежать правильной классификации, является трудной задачей. Противники пытаются нарушить любые предположения, сделанные специалистами-практиками, пока не найдут путь в систему с наименьшим сопротивлением.

Большой класс уязвимостей методов машинного обучения происходит от фундаментальной проблемы несовершенного обучения. Алгоритм машинного обуче-

ния пытается подогнать гипотетическую функцию, которая отображает точки, взятые из конкретного пространства распределения данных, в различные категории или в некоторый числовой диапазон. В качестве простого мысленного эксперимента предположим, что необходимо тренировать статистически обучаемого агента для распознавания атак межсайтового скриптинга (XSS)[1] в веб-приложениях. Идеальным результатом является агент, способный выявлять во входных данных любые возможные вариации XSS с абсолютной точностью и без ложноположительных результатов. В действительности невозможно создать систему с абсолютной эффективностью, которая успешно решает сложные задачи, потому что обучаемого агента невозможно обеспечить абсолютно точной и полной информацией. Нет никакой возможности предоставить обучаемому набор данных, извлеченный из полного распределения исходных данных обо всех возможных вариациях XSS. Следовательно, существует некоторый сегмент распределения, который мы предназначаем для передачи обучаемому, но в действительности этот сегмент не дает достаточной информации о предмете обучения. Ошибка моделирования (modeling error) – другое явление, которое вносит свой вклад во враждебное пространство, окружающее статистического обучаемого. Статистическое обучение формирует абстрактные модели, которые описывают реальные данные, а ошибки моделирования возникают из-за естественных недостатков, присущих моделям, сформированных таким способом.

Даже «идеальный обучаемый» может допускать уязвимости, поскольку коэффициент ошибок Байеса[2] может быть ненулевым. Коэффициент ошибок Байеса – это нижняя граница возможной ошибки для заданной комбинации статистического классификатора и используемого набора признаков. Этот коэффициент ошибок полезен для числовой оценки качества набора признаков, а также для измерения эффективности классификатора. Коэффициент ошибок Байеса представляет теоретический предел эффективности классификатора. Это означает, что даже если предоставить классификатору полное представление данных, устраняющее все источники несовершенного обучения, все равно будут существовать конечные наборы «неправильных» экземпляров, которые могут привести к некорректной классификации.

На рис. 8.1 показана теоретическая совокупность данных, необходимая для разработки статистической модели обучения, и ее взаимосвязи с пространствами распределения тренировочного и тестового наборов данных. (Отметим, что здесь речь идет не о реальных наборах данных, а о совокупности, из которых эти наборы извлекаются, – в действительности не должно быть пересечения между тренировочным и тестовым наборами данных.)

По существу, тренировочные данные, предоставляемые алгоритму машинного обучения, выбираются из неполного сегмента теоретического пространства распределения. Когда наступает время оценки модели в лабораторных условиях или в реальной практике, тестовый набор (взятый из распределения тесто-

---

[1] Атаки межсайтового скриптинга (XSS) обычно используют уязвимости веб-приложений, которые позволяют атакующим внедрить скрипты клиентской стороны в веб-страницы, просматриваемые другими пользователями.

[2] *Keinosuke Fukunaga*. Introduction to Statistical Pattern Recognition. 2nd ed. (San Diego, CA: Academic Press 1990), pp. 3 and 97.

вых данных) может содержать сегмент данных, свойства которых не захвачены в распределении тренировочных данных. Этот сегмент мы обозначаем термином «враждебное пространство» (adversarial space). Атакующие могут воспользоваться зонами враждебных пространств между множествами данных, адаптированными для статистически обучаемого агента, и теоретическим пространством распределения, чтобы обмануть алгоритмы машинного обучения. Специалисты-практики по машинному обучению и проектировщики систем ожидают, что тренировочный и тестовый наборы данных должны извлекаться из того же пространства распределения, и делают дальнейшее предположение о том, что все характеристики этого теоретического распределения должны покрываться тренируемой моделью. Такие «слепые пятна» в алгоритмах машинного обучения возникают из-за расхождений между ожиданиями и действительностью.

Рис. 8.1 ❖ Враждебное пространство как результат несовершенного представления тренировочных данных

Но еще бóльшая опасность возникает, когда атакующие получают возможность воздействовать на процесс тренировки и могут сделать недействительными предположения о стабильности данных в процессах машинного обучения. Системы, выполняющие онлайновое обучение (т. е. обучающиеся на основе обратной связи с пользователем в реальном времени), встречаются достаточно часто, так как требования к высокой адаптируемости предполагают получение преимуществ, свойственных таким самоусовершенствующимся статистическим системам. Но онлайновое обучение вводит новый класс уязвимостей, вызывающих заражение моделей, который необходимо принять во внимание.

Источником интеллекта моделей статистического обучения являются предоставляемые им данные, а уязвимости таких систем естественным образом возникают из неадекватности этих данных. Для специалистов-практиков важно убедиться в том, что тренировочные данные являются представлением действительного распределения, заслуживающим доверия в той степени, насколько это

возможно. В то же время необходимо постоянно поддерживать активную защиту и отслеживать различные векторы атак, чтобы иметь возможность проектировать алгоритмы и системы, более устойчивые к атакам.

Мобильность атак

Явление мобильности атак (attack transferability) было открыто исследователями, которые обнаружили, что враждебные экземпляры (взятые из враждебного пространства), которые специально предназначены для возникновения неправильной классификации в одной модели, также с высокой вероятностью приведут к неправильной классификации в других независимо тренируемых моделях[1], – даже если эти две модели основаны на абсолютно различных алгоритмах или инфраструктурах[2]. Это совершенно неочевидное явление с учетом того, например, что функция, используемая методом опорных векторов для распределения тренировочных данных, предположительно имеет лишь небольшое сходство с функцией, используемой глубокой нейронной сетью. С другой стороны, для враждебных пространств во множествах данных тренируемой методами машинного обучения модели A обнаружены значительные пересечения с враждебными пространствами некоторой произвольной модели B.

Мобильность враждебных атак имеет важные последствия для атак, проводимых в действительности на системы машинного обучения, поскольку параметры модели далеко не всегда предъявляются пользователям, взаимодействующим с системой. Исследователи разработали практические враждебные обходные атаки на так называемых моделях «черный ящик» (black-box models), т. е. классификаторы, для которых почти нет информации об используемом методе или модели машинного обучения[3]. При доступе только к тестовым выборкам и результатам, полученным от такого классификатора «черного ящика», можно сгенерировать тренировочный набор данных с метками, с помощью которого можно тренировать локальную заменяющую модель (local substitute model). Затем имеется возможность проанализировать эту локальную заменяющую модель в режиме офлайн для поиска экземпляров, принадлежащих враждебному пространству. Следовательно, явление мобильности атак позволяет воспользоваться этими враждебными экземплярами, чтобы обмануть удаленную модель черного ящика.

Мобильность атак представляет область активных исследований[4]. Это направление исследований будет оказывать непрерывное воздействие на область состязательного машинного обучения в ближайшем обозримом будущем.

---

1 *Christian Szegedy et al*. Intriguing Properties of Neural Networks (2013).
Ian Goodfellow, Jonathon Shlens and Christian Szegedy. Explaining and Harnessing Adversarial Examples (2014).

2 *Nicolas Papernot, Patrick McDaniel and Ian Goodfellow*. Transferability in Machine Learning: From Phenomena to Black-Box Attacks Using Adversarial Samples (2016).

3 *Nicolas Papernot et al*. Practical Black-Box Attacks Against Deep Learning Systems Using Adversarial Examples (2016).

4 *Florian Tramèr et al*. The Space of Transferable Adversarial Examples (2017).

Генеративно-состязательная сеть

Возможно, вам известен класс алгоритмов глубокого обучения под названием генеративно-состязательные сети (generative adversarial networks – GAN)[1]. Это алгоритмы машинного обучения без учителя, в которых используются две нейронные сети-«дуэлянты» в рабочей среде игры с нулевой суммой (zero-sum game) (антагонистической игры).

Обычно одна из двух сетей действует как генератор (generator), а другая – как дискриминатор (discriminator). Дискриминатор тренируется как классификатор по одному классу с помощью обычной методики, принимая экземпляры с метками из тренировочного набора до тех пор, пока не научится правильно прогнозировать принадлежность к классу элементов из тестового набора с некоторым заданным уровнем точности. Затем генератор выполняет итеративные попытки генерации экземпляров с целью заставить дискриминатор воспринимать эти экземпляры как принадлежащие к исходному набору данных. Потом весь процесс можно полностью повторить (итеративно), а завершается он, когда эффективность соответствует системным требованиям или когда дополнительные итерации не улучшают эффективность. Подобный метод тренировки с возвратами (back-and-forth method) позволяет получить систему с необычайно мощными возможностями обучения.

Генеративно-состязательные сети не связаны напрямую с методиками состязательного машинного обучения, но эта технология на практике используется исследователями для генерации имен доменов управления и контроля[2] с целью уклонения от атак в моделях выявления на основе методов машинного обучения.

Методика атак: заражение модели

Атаки с целью заражения (отравления) модели, также известные под названием атаки с ложным отвлекающим маневром (red herring attacks)[3], на практике наблюдаются только в онлайновых системах обучения. Онлайновые системы обучения жертвуют стабильностью ради адаптируемости, достигаемой посредством динамической повторной тренировки моделей машинного обучения с использованием самых актуальных свежих взаимодействий с пользователем или канала обратной связи. Системы выявления аномалий применяют онлайновое обучение для автоматического уточнения параметров модели со временем по мере того, как они выявляют изменения в обычном трафике. При таком подходе можно избежать трудоемкого вмешательства человека в непрерывный процесс точной настройки моделей и уточнения пороговых значений. Но онлайновые обучаемые объекты имеют собственный набор потенциальных опасностей во враждебных

---

1 *Ian Goodfellow et al*. Generative Adversarial Nets. Proceedings of the 27th International Conference on Neural Information Processing Systems (2014): 2672–2680.

2 *Hyrum S. Anderson, Jonathan Woodbridge and Bobby Filar*. DeepDGA: Adversarially-Tuned Domain Generation and Detection. Proceedings of the 2016 ACM Workshop on Artiicial Intelligence and Security (2016): 13–21.

3 Ложный отвлекающий маневр (red herring) – это некоторое действие, которое вводит в заблуждение или отвлекает внимание, – хорошо подходит для атак заражения модели, целью которых является переключение внимания обучаемого агента на обучение в неправильном и/или непредусмотренном направлении.

средах. Особенно в системах, в которых не предусмотрена устойчивость и сопротивляемость атакующим действиям, злоумышленник может без затруднений ввести в заблуждение алгоритмы машинного обучения, внедряя искусственно сформированный трафик.

По определению и по своей природе атаки с целью заражения модели являются каузативными атаками и способны демонстрировать широкое разнообразие с точки зрения специфики и типа нарушения защиты. Рассмотрим пример с сервисом перевода выражений естественного языка со встроенным онлайновым циклом обратной связи с пользователем, который принимает исправления от пользователей непрерывной повторной тренировки механизма перевода на основе машинного обучения. Без какой-либо формы фильтрации вводимых данных беспорядочная атака на эту систему может быть такой же простой, как направление в канал обратной связи бессмысленного «мусора», как показано на рис. 8.2[1]. Более целенаправленная атака может быть проведена на систему при избирательных и многократно повторяемых попытках заставить систему переводить слово love на английском языке как слово déteste на французском языке, как показано на рис. 8.3.

Рис. 8.2 ❖ Беспорядочная попытка заражения системы перевода выражений на естественных языках

Рис. 8.3 ❖ Целенаправленная попытка заражения системы перевода выражений на естественных языках

[1] Снимки экрана сделаны на сервисе Google Translate (**https://translate.google.com/**) и используются только для демонстрации механизма атак рассматриваемого здесь типа. Многие люди полагаются на точность работы онлайновых сервисов, подобных этому, и не следует пытаться сделать нечто подобное без предварительного разрешения, полученного от провайдера сервиса.

Нетрудно понять, как можно отрицательно воздействовать на модель, вводя подобные данные. Мировоззрение статистически обучаемого агента формируется исключительно на основе получаемых им тренировочных данных и любого положительного или отрицательного подкрепления предположений об изучаемом экземпляре. Когда маленький ребенок начинает узнавать названия фруктов по примерам из иллюстрированной книги, процесс обучения точно так же может быть заражен, если изображения фруктов в книге сопровождаются неправильными названиями.

При заражающих атаках предполагается, что атакующие получают управление частью тренировочных данных, используемых алгоритмом обучения. Чем больше пропорциональная доля тренировочных данных, которой управляют атакующие, тем большее воздействие они оказывают на цели обучения и границы решений системы машинного обучения[1]. Атакующий, завладевший управлением более 50 % тренировочного набора, может воздействовать на модель в гораздо большей степени, чем атакующий, который управляет только 5 %. Из этого следует, что более популярные сервисы, для которых наблюдается больший объем корректного трафика, труднее заразить, потому что атакующим придется внедрить гораздо больше помех (chaff)[2], чтобы оказать существенное воздействие на итоговые результаты обучения.

Разумеется, владельцы систем могут с легкостью обнаружить тот факт, что онлайновый обучаемый объект внезапно начинает получать слишком большой объем загрязненных тренировочных данных. Простые правила могут помочь выявить и пометить случаи неожиданных пиков подозрительного или аномального поведения, которые могут означать вредоносные и/или мошеннические действия. После обнаружения таких событий фильтрация трафика становится достаточно простой задачей. Так называемые атаки типа «лягушка в кипятке» (boiling frog attacks) распределяют действия по внедрению некорректных экземпляров обучающих данных в течение значительного интервала времени, поэтому не генерируются какие-либо сигналы об опасности. Атаки типа «лягушка в кипятке» можно сделать более эффективными и менее подозрительными, вводя вредоносный трафик постепенно по этапам, соответствующим постепенным сдвигам границы решения классификатора.

Заражающие атаки, выполняемые последовательно в течение долгого времени, можно организовать так, чтобы они выглядели как естественное изменение распределений данных. Например, детектор аномалий в режиме онлайнового обучения, граница решения которого изначально была установлена для блокировки, начинающейся с 10 запросов в минуту (или с одного IP-адреса), должен блокировать запросы с IP-адресов, генерирующих 20 запросов в минуту. Такая система вряд ли была сконфигурирована для обучения по новому трафику, так как детектор должен классифицировать его как аномалию с высокой степенью уверенности (достоверности). Но слишком долгое «зависание» в непосредственной близости от границы решения может привести к тому, что системы такого

---

[1] Предполагается, что все экземпляры, используемые для тренировки модели, имеют равные весовые коэффициенты и вносят одинаковый вклад в процесс тренировки модели.

[2] Chaff – специальный термин, обозначающий трафик атаки, предназначенный для заражения (отравления, загрязнения) моделей машинного обучения.

типа «пересматривают» изначально выбранные и подогнанные гипотетические функции. Атакующий, который начинает с отправки 11 запросов в минуту в течение одной недели, получает более высокий шанс сдвинуть границу решения с 10 до 11. Повторение этого процесса с новой границей решения может помочь в достижении главной цели – значительного изменения границы решения без генерации каких-либо предупреждений об опасности. Для системных администраторов может существовать множество разнообразных вполне объяснимых (с их точки зрения) причин такого смещения: возрастающая популярность веб-сайта, рост удерживающего интереса пользователей, ведущий к более длительным взаимодействиям, увеличение потока новых пользователей и т. д.

Заражающие атаки продолжают исследоваться и демонстрироваться с использованием разнообразных методов машинного обучения и практически применяемых систем: методов опорных векторов[1], алгоритмов выявления аномалий с использованием центроидов и общих точек[2], логистической, линейной и гребневой регрессий[3], фильтров спама[4], классификаторов вредоносного ПО[5], процессов выбора признаков[6], метода главных компонентов (PCA)[7] и алгоритмов глубокого обучения[8].

Пример: заражающая атака на бинарный классификатор

Для наглядной демонстрации заражающей атаки рассмотрим конкретный пример воздействия атакующего на границу решения простого классификатора на основе машинного обучения. Для атакующего не ограничены запросы доступа к прогнозам системы[9]. Начнем с создания случайно сгенерированного (искусственного) набора данных с помощью утилиты `sklearn.datasets.make_classification()` (**https://scikit-learn.org/stable/modules/generated/sklearn.datasets.make_classification.html**):

---

[1] *Battista Biggio, Blaine Nelson and Pavel Laskov*. Poisoning Attacks Against Support Vector Machines. Proceedings of the 29th International Conference on Machine Learning (2012): 1467–1474.

[2] *Marius Kloft and Pavel Laskov*. Security Analysis of Online Centroid Anomaly Detection. Journal of Machine Learning Research 13 (2012): 3647–3690.
Benjamin I. P. Rubinstein et al. ANTIDOTE: Understanding and Defending Against Poisoning of Anomaly Detectors. Proceedings of the 9th ACM SIGCOMM Internet Measurement Conference (2009): 1–14.

[3] *Shike Mei and Xiaojin Zhu*. Using Machine Teaching to Identify Optimal Training-Set Attacks on Machine Learners. Proceedings of the 29th AAAI Conference on Artiicial Intelligence (2015): 2871–2877.

[4] *Blaine Nelson et al*. Exploiting Machine Learning to Subvert Your Spam Filter. Proceedings of the 2nd USENIX Workshop on Large-Scale Exploits and Emergent hreats (2008): 1–9.

[5] *Battista Biggio et al*. Poisoning Behavioral Malware Clustering. Proceedings of the 7th ACM Workshop on Artiicial Intelligence and Security (2014): 27–36.

[6] *Huang Xiao et al*. Is Feature Selection Secure Against Training Data Poisoning? Proceedings of the 32nd International Conference on Machine Learning (2015): 1689–1698.

[7] *Ling Huang et al*. Adversarial machine learning. Proceedings of the 4th ACM Workshop on Artiicial Intelligence and Security (2011): 43–58.

[8] *Luis Muñoz-González et al*. Towards Poisoning of Deep Learning Algorithms with Back-Gradient Optimization. Proceedings of the 10th ACM Workshop on Artiicial Intelligence and Security (2017): 27–38.

[9] Полный код этого примера в виде Python Jupyter notebook *chapter8/binary-classifier-evasion.ipynb* (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter8/binary-classifier-evasion.ipynb**) в репозитории кода для данной книги.

```
from sklearn.datasets import make_classification

X, y = make_classification(n_samples=200,
    n_features=2,
    n_informative=2,
    n_redundant=0,
    weights=[.5, .5],
    random_state=17)
```

Этот код создает простой набор данных с двумя признаками, состоящий из 200 элементов, из которого первые 100 элементов будут выбраны для тренировочного набора, а остальные 100 элементов – для визуальной демонстрации того, что классификатор натренирован должным образом.

Для этого примера выбран классификатор типа многослойный перцептрон (MLP), подходящий к исследуемому набору данных[1]. Многослойные перцептроны – это класс простых нейронных сетей с прямой связью, которые могут создавать нелинейные границы решений. В рассматриваемом примере импортируется класс `sklearn.neural_network.MLPClassifier` (**https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.html**), и соответствующая модель подгоняется к исследуемому набору данных:

```
from sklearn.neural_network import MLPClassifier

clf = MLPClassifier(max_iter=600, random_state=123).fit(X[:100], y[:100])
```

Для наблюдения за тем, что происходит внутри модели, сгенерируем визуальное представление функции принятия решения классификатором. Создается двумерная сеть-решетка точек в пространстве исследуемых входных данных (значения X и y между –3 и 3 с интервалами 0,01 между каждой парой смежных точек), затем извлекаются вероятности прогнозов для каждой точки в этой сети-решетке:

```
import numpy as np

xx, yy = np.mgrid[-3:3:.01, -3:3:.01]
grid = np.c_[xx.ravel(), yy.ravel()]
probs = clf.predict_proba(grid)[:, 1].reshape(xx.shape)
```

После этого генерируется контурный график (схема) по полученной информации и на него накладывается тестовый набор данных с отображением X_1 по вертикальной оси и X_0 по горизонтальной оси:

```
import matplotlib.pyplot as plt

f, ax = plt.subplots(figsize=(12, 9))

# Построение контура как нижнего слоя (фона)
contour = ax.contourf(xx, yy, probs, 25, cmap="RdBu",
                      vmin=0, vmax=1)
ax_c = f.colorbar(contour)
ax_c.set_label("$P(y = 1)$")
```

[1] Отметим, что в данном случае выбор классификатора был произвольным. Многослойный перцептрон здесь используется, потому что он является одним из классификаторов, для которых реализована функция `partial_fit()`, в дальнейшем применяемая нами для имитации постепенной тренировки модели, принятой для онлайновых обучаемых объектов.

```
ax_c.set_ticks([0, .25, .5, .75, 1])

# Отображение тестового набора (второй половины набора данных X и y)
ax.scatter(X[100:,0], X[100:, 1], c=y[100:], s=50,
           cmap="RdBu", vmin=-.2, vmax=1.2,
           edgecolor="white", linewidth=1)

ax.set(aspect="equal",
       xlim=(-3, 3), ylim=(-3, 3))
```

На рис. 8.4 показан результат.

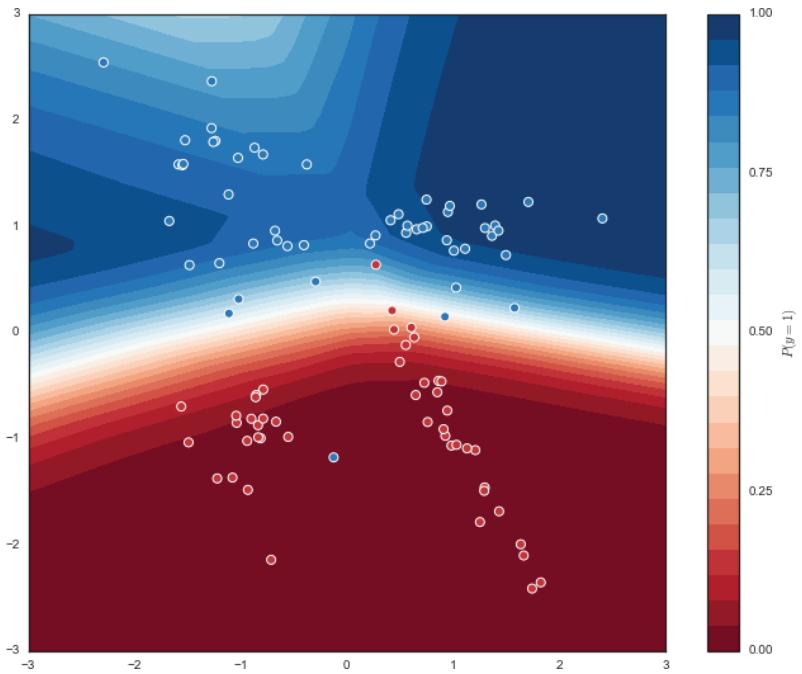


Рис. 8.4 ❖ Графическое изображение контура функции решения классификатора MLP, подогнанного к исследуемому набору данных

На рис. 8.4 показано, что функция решения классификатора MLP выглядит вполне подходящей (хорошо подогнанной) к тестовому набору данных. В качестве границы решения здесь используется пороговое значение достоверности 0,5. Таким образом, если прогнозы классификатора с P(y = 1) > 0,5, то прогноз y = 1, в противном случае прогноз y = 0. Также определяется вспомогательная функция plot_decision_boundary() для графического построения этой границы решения вместе с тем же тестовым набором данных[1]:

```
plot_decision_boundary(X, y, probs)
```

[1] Полный код реализации этой функции можно найти в *chapter8/binary-classifier-evasion.ipynb* (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter8/binary-classifier-evasion.ipynb**) в репозитории кода для данной книги. Приводим полную сигнатуру этой функции: `plot_decision_boundary(X_orig, y_orig, probs_orig, chaff_X=None, chaff_y=None, probs_poisoned=None)`.

Результат показан на рис. 8.5.

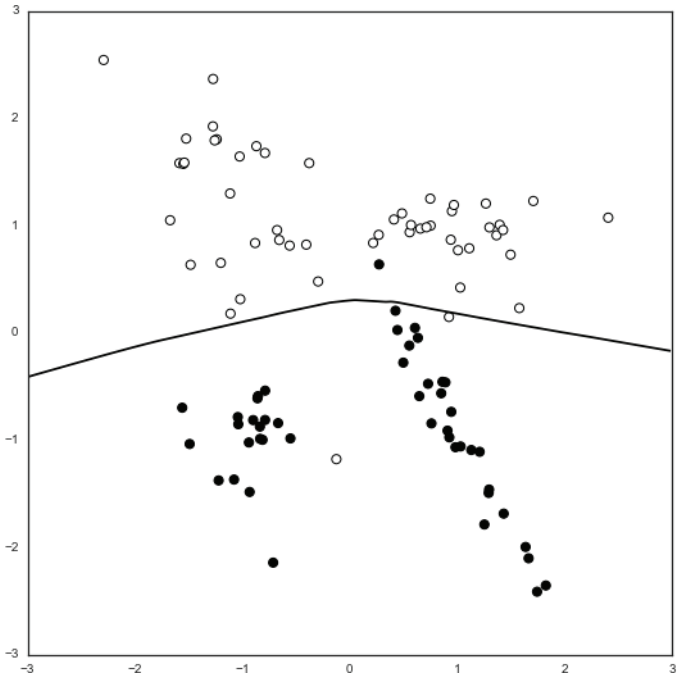


Рис. 8.5 ❖ Граница решения классификатора MLP, подогнанная под исследуемый набор данных

Далее генерируются пять тщательно выбранных точек помех (chaff), представляющих только 5 % тренировочного набора данных. Этим точкам присваивается метка y = 1, поскольку именно это значение должен спрогнозировать классификатор (с учетом текущей функции решения):

```
num_chaff = 5
chaff_X = np.array([np.linspace(-2, -1, num_chaff),
    np.linspace(0.1, 0.1, num_chaff)]).T
chaff_y = np.ones(num_chaff)
```

На рис. 8.6 показаны точки помех (обозначенные маркерами-звездочками), которые в основном расположены в пространстве y = 1 (пространство y = 1 обозначено пустыми (белыми) маркерами-точками, а пространство y = 0 – закрашенными (черными) маркерами-точками).

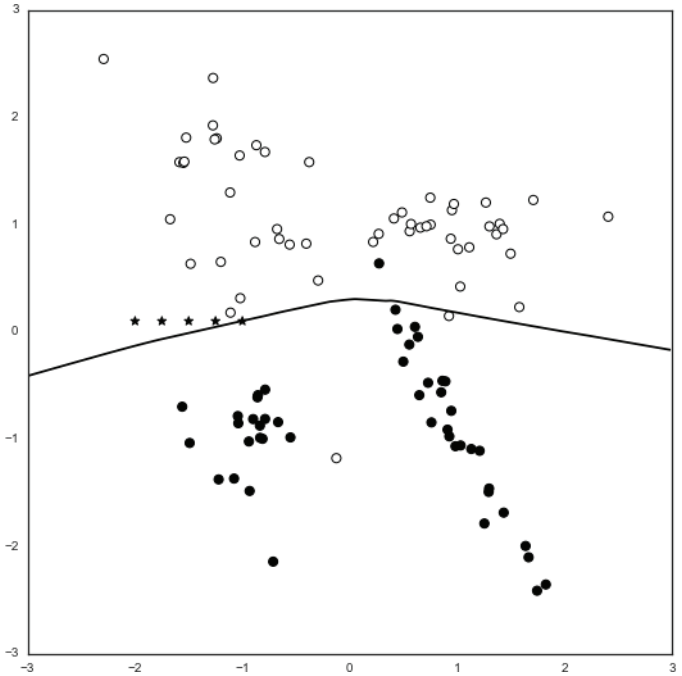


Рис. 8.6 ❖ Точки помех (chaff), отображенные по отношению к тестовому набору

Для имитации онлайнового обучаемого объекта, использующего новые принимаемые данные в динамическом режиме для постепенной тренировки модели машинного обучения, воспользуемся API `partial_fit()` библиотеки scikit-learn для постепенного обучения, где реализованы некоторые механизмы оценки (включая `MLPClassifier`). Затем выполняется постепенная тренировка существующего классификатора с помощью частичной подгонки модели к пяти новым точкам помех (трафик атаки), сгенерированным ранее:

```
clf.partial_fit(chaff_X, chaff_y)
```

Теперь классификатор обновлен с помощью свежей информации о вредоносных действиях. Далее наблюдаем, как была сдвинута граница решения (см. рис. 8.7):

```
probs_poisoned = clf.predict_proba(grid)[:, 1].reshape(xx.shape)
plot_decision_boundary(X, y, probs, chaff_X, chaff_y, probs_poisoned)
```

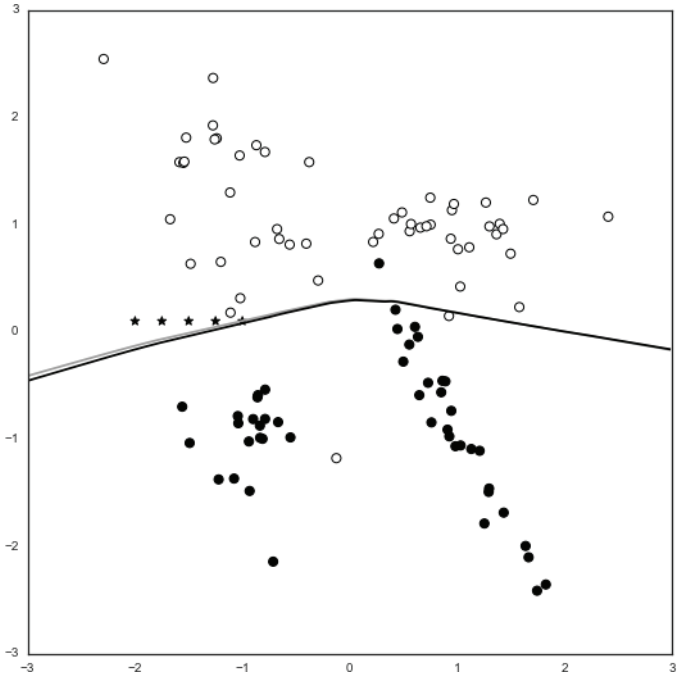


Рис. 8.7 ❖ Сдвинутая граница решения
после 1x частичной подгонки по пяти точкам помех

Новая граница решения показана более темной линией. Отметим, что граница решения была незначительно смещена вниз, создавая очень маленькое пространство между двумя кривыми. Любые точки, лежащие в этом пространстве, ранее должны были классифицироваться как y = 0, но теперь такие точки должны классифицироваться как y = 1. Это означает, что атакующий добился успеха в достижении своей цели – неправильной классификации экземпляров. Итеративное повторение шага частичной подгонки `partial_fit()` (с использованием тех же пяти точек помех) позволяет наблюдать увеличение сдвига функции (границы) решения по мере роста процентного содержания помех в трафике, как показано на рис. 8.8.

Увеличение смещения границы решения предоставляет более широкое пространство входных данных для неправильно классифицируемых точек, что приводит к серьезной деградации эффективности модели.

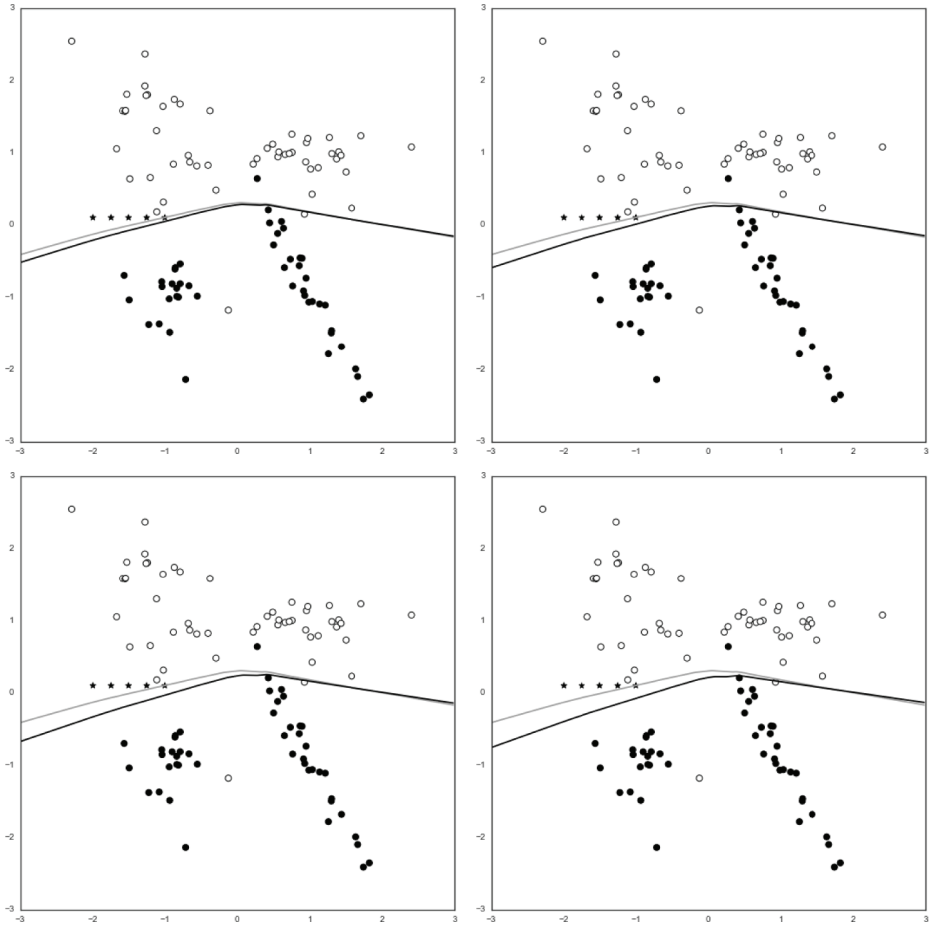


Рис. 8.8 ❖ Постепенный сдвиг границы решения после 2×, 3×, 4× и 5× операций частичной подгонки по пяти точкам помех (соответственно 10 %, 15 %, 20 %, 25 % трафика атаки, начиная с левого верхнего графика до правого нижнего)

Знания атакующего

Как можно заметить на рис. 8.8, знание внутренней функции решения модели является важным фактором заражающих атак. Каким образом атакующий может узнать, как правильно выбрать помехи таким способом, чтобы вызвать требуемое смещение границы решения?

Базовый уровень доступа, который, по нашему предположению, предоставляется атакующему, дает ему возможность выполнить неограниченное количество запросов к системе и получить результаты прогнозов. Известно, что чем меньше запросов обращено к системе, тем менее вероятно, что атакующий вызовет подозрение при срабатывании системы оповещения об опасности. Атакующий, получивший доступ не только к результатам прогнозов, но и к данным о вероятностях

прогнозов, занимает гораздо более выгодную позицию, так как может определить градиенты функции решений (например, как показано на рис. 8.4), позволяющие провести требуемые операции оптимизации, особенно при выборе точек помех на сложных поверхностях функции решения, где эта функция имеет многочисленные локальные минимумы и максимумы.

Но даже если атакующий, не имеющий возможности узнать вероятности прогнозов, получает доступ к результатам категориальной классификации, они позволят ему определить границы решения модели с помощью выполнения запросов по точкам, расположенным в непосредственной близости к границе (как показано на рис. 8.5–8.8). Этой информации достаточно для атакующего, чтобы правильно выбрать трафик помех, которые не вызовут подозрений, но могут сбить с толку онлайновый обучаемый объект.

Одним из вариантов, при котором определение размещения помех требует применения реверс-инженерии системы машинного обучения, является преобразование входных данных перед передачей их в классификатор. Например, если метод снижения размерности PCA применяется к данным, то как атакующий может узнать, какие именно размерности обрабатываются? Аналогично, если входные данные подвергаются некоторому другому неизвестному нелинейному преобразованию до передачи в классификатор, то еще менее понятно, как атакующий должен найти связь изменений в исходных данных с точками на поверхности решения. Еще один пример – некоторые типы пользовательских входных данных не могут быть без затруднений преобразованы пользователем, взаимодействующим с системой, например если входные данные для классификатора зависят от состояния системы или от свойств, на которые пользователи не могут повлиять. Наконец, правильное определение количества помех, необходимого для требуемого смещения границы решения, также может оказаться весьма трудной задачей.

Почти все перечисленные выше трудности можно преодолеть при достаточном уровне доступа к системе, позволяющем атакующему извлекать всю возможную информацию из обучаемого объекта. Цель защищающейся стороны – без потери эффективности максимально затруднить для атакующих выполнение простых действий, влияющих на систему и позволяющих проводить атаки на обучаемый объект.

Защита от заражающих атак

Существует несколько проектных решений, которые архитектор системы машинного обучения может применить для затруднения действий по заражению модели даже для мотивированных противников. Должна ли система действительно работать в реальном времени с обеспечением возможности ежеминутного онлайнового обучения или почти такой же результат можно получить с помощью ежедневно планируемой постепенной тренировки с использованием данных предыдущего дня? Проектирование онлайновых систем обучения, поведение которых похоже на поведение системы поэтапного пакетного обновления в режиме офлайн, дает следующие значительные преимущества:

- более длительные интервалы времени между повторными тренировками предоставляют системе возможность инспекции (проверки) данных, передаваемых в модель;

- анализ данных за продолжительный интервал времени повышает вероятность выявления атак типа «лягушка в кипятке», когда помехи внедряются постепенно в течение длительного времени. Объединение данных, собранных за неделю, в противоположность данным, полученным в течение последних пяти минут, позволяет выявлять помехи, внедряемые с низкой скоростью;
- атакующие добиваются успеха в коротких циклах обратной связи. Заметное смещение границы решения как результат генерируемого атакующими трафика атаки дает им быстрое положительное подкрепление и позволяет продолжить многократное применение выбранного метода. Организация цикла обновления один раз в неделю означает, что атакующие не будут знать, была ли успешной их попытка атаки, поскольку им придется ждать результатов до следующей недели.

Если имеется механизм для инспектирования набора данных для постепенной тренировки перед их передачей в алгоритм частичного обучения, то можно предложить несколько способов выявления попыток заражающих атак:

- идентификация аномальных пакетов в трафике, исходящем с одного IP-адреса или автономной системы (ASN), или обнаружения необычных общих характеристик. Например, множество входящих запросов с нестандартной строкой юзер-агента. Можно удалить такой трафик из данных для автоматического механизма постепенного обучения и поручить аналитику исследование этого трафика для определения признаков атак;
- поддержка калибровочного набора созданного вручную тестового набора данных «обычного трафика», который используется для проверки модели после каждого этапа повторной тренировки. Если результаты классификации в текущем цикле значительно отличаются от результатов предыдущих циклов, то, возможно, была предпринята попытка атаки;
- определение порогового значения вблизи границы решения и постоянное наблюдение за процентной долей точек тестовых данных, попадающих в это пространство. Например, предположим, что имеется простая линейная граница решения $P(y = 1) = 0{,}5$. Если определить область порогового значения у границы решения как $0{,}4 < P(y = 1) < 0{,}6$, то можно вычислить процентную долю точек тестовых данных, которые по ежедневным наблюдениям попадают в эту область достоверности прогнозов. Например, если в среднем 30 % точек обычно наблюдается в этой области, но за прошедшую неделю происходит резкий рост количества точек в области порогового значения до 80 %, то подобная аномалия может быть признаком заражающей атаки – атакующие пытаются максимально приблизиться к границе решения, чтобы получить возможность сместить эту границу без возникновения сигналов об опасности.

Из-за большого разнообразия атак заражения модели, направленных на системы машинного обучения, невозможно сформулировать универсальное быстродействующее правило создания абсолютной защиты системы от атак такого типа. Это область активных иссследований, в этом направлении продолжаются алгоритмические разработки методов статистического обучения, которые в меньшей степени уязвимы для заражающих атак. Робастную статистику (robust statistics)

часто упоминают как потенциально возможное решение для создания алгоритмов, более устойчивых к вредоносным обманным действиям[1].

Методика атаки: искажающая атака

Использование враждебного пространства (показанного на рис. 8.1) для поиска враждебных (аномальных) экземпляров, которые приводят к неправильной классификации в классификаторе на основе машинного обучения, называют искажающей атакой (evasion attack). Средства массовой информации сравнили это явление с оптическими иллюзиями, обманывающими людей, главным образом из-за того, что первоначальные исследования в этой области демонстрировали основную концепцию с помощью классификаторов изображений на основе глубокой нейронной сети[2]. Например, на рис. 8.9 показано, как незначительные изменения, внесенные в интенсивность отдельных пикселов, могут привести высокоэффективный классификатор цифр MNIST к неправильной классификации 0 как 6.

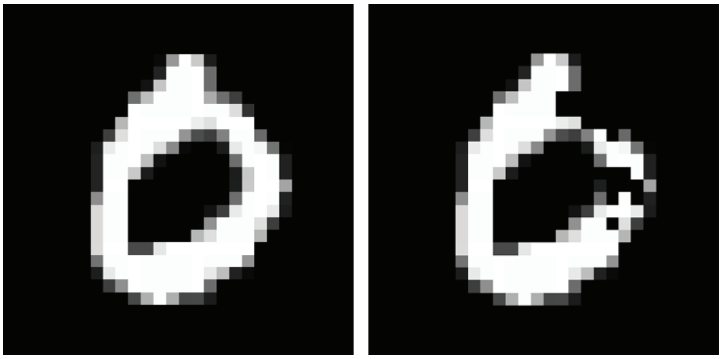


Рис. 8.9 ❖ Сравнение исходного неизмененного изображения рукописной цифры 0 из набора MNIST (слева) с преднамеренно искаженной версией изображения (справа)

Искажающие атаки заслуживают внимания, потому что они более легки в применении, чем заражающие атаки. Прежде всего подобные атаки могут воздействовать на любой классификатор, даже если у пользователя нет возможности как-либо воздействовать на этап тренировки. В сочетании с явлением мобильности атак и тренировочным процессом локальной заменяющей модели разведывательно-исследовательская природа этой методики означает, что мотивирован-

---

[1] *Emmanuel J. Candès et al*. Robust Principal Component Analysis? (2009).
Mia Hubert, Peter Rousseeuw and Karlien Vanden Branden. ROBPCA: A New Approach to Robust Principal Component Analysis. Technometrics 47 (2005): 64–79.
S. Charles Brubaker. Robust PCA and Clustering in Noisy Mixtures. Proceedings of the 20th Annual ACM-SIAM Symposium on Discrete Algorithms (2009): 1078–1087.
Peter Rousseeuw and Mia Hubert. Anomaly Detection by Robust Statistics (2017).
Sohil Atul Shah and Vladlen Koltun. Robust Continuous Clustering. Proceedings of the National Academy of Sciences 114 (2017): 9814–9819.

[2] *Ian Goodfellow, Jonathon Shlens and Christian Szegedy*. Explaining and Harnessing Adversarial Examples. ICLR 2015 conference paper (2015).

ные атакующие противники могут осуществлять вполне целенаправленные атаки на целостность большого класса систем машинного обучения. Искажающие атаки с вредоносными экземплярами продемонстрировали возможности сильного воздействия как на обычные модели машинного обучения (логистическая регрессия, методы опорных векторов, метод поиска ближайших соседей, деревья решений и т. д.), так и на модели глубокого обучения.

Исследователи также продемонстрировали, что случаи неправильной классификации изображений могут приводить к значительным последствиям в реальной жизни[1]. В частности, разработка автономных средств передвижения привела исследователей к поиску состязательных экземпляров данных, устойчивых к случайному шуму и случайным преобразованиям[2]. Исследователи показали, как чрезвычайно малые искажения в дорожных знаках могут привести к неправильной классификации изображений в автомобилях с автоматическим управлением[3].

Разумеется, цели атак этого типа не ограничиваются системами классификации изображений. Исследователи продемонстрировали аналогичную атаку с успешным обходом классификаторов вредоносного ПО[4]. Прямым следствием этой демонстрации стала уверенность специалистов в области обеспечения безопасности в том, что машинное обучение является главным компонентом механизма выявления угроз. Вредоносные искажения удобны для применения в процессе распознавания образов, потому что изменение нескольких пикселов изображения обычно не приводит к каким-либо заметным визуальным эффектам. С другой стороны, применение той же концепции к выполняемым бинарным файлам требует некоторых навыков хакерства (взлома) и экспериментов, чтобы убедиться в том, что измененные биты (или программные инструкции, строки кода и т. п.) не приводят к потере работоспособности бинарного файла или к потере его изначальных вредоносных свойств и поведения, которые могут воспрепятствовать достижению главной цели искажающей атаки.

Пример: искажающая атака на бинарный классификатор

Продемонстрируем основные принципы искажающих атак с помощью попытки поиска вредоносного экземпляра данных с использованием алгоритма элементарного градиентного подъема. Предположим, что хорошо осведомленный атакующий противник имеет полный доступ к тренируемой модели машинного обучения.

1. Начинаем с произвольно выбранного экземпляра выборки, для которого имеется модель генерации вероятностей прогнозов.
2. Модель анализируется для поиска признаков, обладающих наибольшими весовыми коэффициентами в направлении, в котором должна происходить неправильная классификация, т. е. мы ищем признак J, который заставляет классификатор быть менее уверенным в его исходном прогнозе.

---

[1] *Alexey Kurakin, Ian Goodfellow and Samy Bengio*. Adversarial Examples in the Physical World (2016).

[2] *Anish Athalye et al*. Synthesizing Robust Adversarial Examples (2017).

[3] *Ivan Evtimov et al*. Robust Physical-World Attacks on Machine Learning Models (2017).

[4] *Weilin Xu, Yanjun Qi and David Evans*. Automatically Evading Classifiers: A Case Study on PDF Malware Classifiers. Proceedings of the 23rd Network and Distributed Systems Symposium (2016).

3. Итеративно увеличивается величина этого признака до тех пор, пока вероятность прогноза не пересечет границу (пороговое значение) достоверности (обычно равную 0,5).

Предварительно тренируемая модель машинного обучения, используемая в этом примере, работает в простом защитном экране веб-приложений (WAF)[1]. Этот защитный экран представляет собой специализированный классификатор XSS. Принимая на входе строку, WAF выдает прогноз, является ли эта строка экземпляром XSS. Приступим к работе.

Хотя в реальной практике атакующие не обладают исчерпывающими знаниями, предположение о том, что в рассматриваемом примере атакующий полностью осведомлен, позволит вычислить наихудшую оценку уязвимостей модели и продемонстрировать максимальные возможности атак этого типа. В рассматриваемом примере предполагается, что атакующий имеет доступ к сериализованному объекту `Pipeline` (**https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html**) из библиотеки scikit-learn и может просматривать каждый этап в конвейере модели.

Во-первых, загружается тренируемая модель и определяется, на каких этапах конвейера используется встроенная функция языка Python vars() (**https://docs.python.org/3/library/functions.html#vars**)[2]:

```
import pickle

p = pickle.load(open('waf/trained_waf_model'))
vars(p)

> {'steps': [
        ('vectorizer',
         TfidfVectorizer(analyzer='char', binary=False,
                         decode_error=u'strict',
                         dtype=<type 'numpy.int64'>,
                         encoding=u'utf-8', input=u'content',
                         lowercase=True, max_df=1.0,
                         max_features=None, min_df=0.0,
                         ngram_range=(1, 3), norm=u'l2',
                         preprocessor=None, smooth_idf=True,
                         stop_words=None, strip_accents=None,
                         sublinear_tf=True,
                         token_pattern=u'(?u)\\b\\w\\w+\\b',
                         tokenizer=None, use_idf=True,
                         vocabulary=None)
        ),
        ('classifier',
         LogisticRegression(C=1.0, class_weight='balanced',
                            dual=False, fit_intercept=True,
                            intercept_scaling=1, max_iter=100,
```

[1] Исходный код и небольшие наборы данных для тренировки и использования WAF можно найти в подкаталоге *chapter8/waf* (**https://github.com/oreilly-mlsec/book-resources/tree/master/chapter8/waf**) в репозитории кода для данной книги.

[2] Полный код этого примера можно найти в Python Jupyter notebook *chapter8/binary-classifier-evasion.ipynb* (**https://github.com/oreilly-mlsec/book-resources/blob/master/chapter8/binary-classifier-evasion.ipynb**) в репозитории кода для данной книги.

```
                        multi_class='ovr', n_jobs=1,
                        penalty='l2', random_state=None,
                        solver='liblinear', tol=0.0001,
                        verbose=0, warm_start=False)
    )]}
```

Здесь видно, что объект `Pipeline` содержит только два этапа: `TfidfVectorizer`, за которым следует классификатор `LogisticRegression`. В этом случае успешный вредоносный экземпляр должен привести к ложноотрицательному результату, т. е. должен быть строкой, содержащей реальный код XSS, но классификатор определит его как обычную корректную строку.

Освежив в памяти ранее полученные знания о векторизаторах текста, понимаем, что теперь необходимо найти конкретные элементы (tokens) строки, которые могут помочь в оказании максимального воздействия на классификатор. Можно проверить словарь элементов векторизатора, просматривая его атрибут `vocabulary_`:

```
vec = p.steps[0][1]
vec.vocabulary_
```

```
> {u'\x00\x02': 7,
   u'\x00': 0,
   u'\x00\x00': 1,
   u'\x00\x00\x00': 2,
   u'\x00\x00\x02': 3,
   u'q-1': 73854,
   u'q-0': 73853,
   ...
  }
```

Каждый из этих элементов связан с весовым коэффициентом частоты слова (term weight) (обратная частота документа – inverse document frequency, TF-IDF, вектор), который передается в классификатор как отдельный признак документа. Тренируемый классификатор `LogisticRegression` содержит коэффициенты, к которым можно получить доступ через атрибут `coef_`. Проверим оба этих массива, чтобы увидеть, как можно ими воспользоваться:

```
clf = p.steps[1][1]
```

```
print(vec.idf_)
```

```
> [  9.88191796  13.29416517  13.98731235  ...,
    14.39277746  14.39277746  14.39277746]
```

```
print(clf.coef_)
```

```
> [[  3.86345441e+00   2.97867212e-02   1.67598454e-03 ...,
      5.48339628e-06   5.48339628e-06   5.48339628e-06]]
```

Произведение векторов весовых коэффициентов частоты слов TF-IDF и коэффициентов `LogisticRegression` точно определяет, какое воздействие каждое слово (term) оказывает на общие вероятности прогнозов:

```
term_influence = vec.idf_ * clf.coef_
print(term_influence)
```

```
> [[  3.81783395e+01   3.95989592e-01   2.34425193e-02 ...,
      7.89213024e-05   7.89213024e-05   7.89213024e-05]]
```

Теперь необходимо ранжировать слова по значению их воздействия. Функция `numpy-argpartition()` (**https://docs.scipy.org/doc/numpy/reference/generated/numpy.argpartition.html**) используется для сортировки массива и преобразования значений в индексы массива `vec.idf_`, и теперь можно найти соответствующую строку в словаре элементов векторизатора `vec.vocabulary_`:

```
print(np.argpartition(term_influence, 1))

> [[81937 92199     2 ..., 97829 97830 97831]]
```

Можно видеть, что элемент с индексом 80832 оказывает наибольшее положительное воздействие на достоверность прогноза. Рассмотрим эту строку подробнее, для чего извлечем ее из словаря элементов:

```
# Сначала создается словарь набора элементов, после чего можно получать доступ
# к элементам по индексу
vocab = dict([(v,k) for k,v in vec.vocabulary_.items()])

# Затем можно внимательно рассмотреть элемент с индексом 80832
print(vocab[81937])
> t/s
```

Добавление этого элемента в содержимое (в полезную нагрузку) входных данных XSS должно заставить классификатор стать чуть менее уверенным в своем прогнозе. Попробуем подобрать произвольную полезную нагрузку (содержимое) и проверим результат, чтобы убедиться в том, что классификатор действительно правильно классифицирует это содержимое как строку XSS (y = 1):

```
payload = "<script>alert(1)</script>"

p.predict([payload])[0]

# Классификатор правильно дает прогноз о том, что это содержимое типа XSS
> 1

p.predict_proba([payload])[0]

# Классификатор на 99.9999997 % уверен в этом прогнозе
> array([  1.86163618e-09,   9.99999998e-01])
```

Далее понаблюдаем, как добавление строки `"t/s"` во входные данные воздействует на вероятность прогноза:

```
p.predict_proba([payload + '/' + vocab[80832]])[0]
> array([  1.83734699e-07,   9.99999816e-01])
```

Уверенность в прогнозе снизилась с 99,9999998 % до 99,9999816 %. Теперь необходимо лишь увеличить весовой коэффициент этого признака в рассматриваемой выборке. Для `TfidfVectorizer` это просто означает увеличение числа повторных появлений соответствующего элемента в вводимой строке. Продолжая увеличивать весовой коэффициент этого признака в рассматриваемой выборке, мы поднимаем градиент уверенности классификатора (достоверности) в правильности целевого класса, т. е. классификатор обретает все большую и большую уверенность в том, что этот экземпляр не является XSS.

В конце концов, мы достигаем точки, в которой вероятность прогноза для класса y = 0 превосходит вероятность прогноза для класса y = 1:

```
p.predict_proba([payload + '/' + vocab[80832]*258])[0]
```

```
> array([ 0.50142443,  0.49857557])
```

И классификатор выдает прогноз: эта входная строка не является строкой XSS:

```
p.predict([payload + '/' + vocab[80832]*258])[0]
```

```
> 0
```

Исследуя эту строку, мы убеждаемся в том, что она несомненно является частью содержимого атаки межсайтового скриптинга:

```
print(payload + '/' + vocab[80832]*258)
```

```
# Выводимый результат сокращен для экономии места
> <script>alert(1)</script>/t/st/st/st/st/st/st/st/st/s...t/s
```

Таким образом, мы успешно нашли вредоносный экземпляр, который обманывает используемый в этом примере защитный экран веб-приложений (WAF).

Продемонстрированная здесь методика работает для чрезвычайно простой линейной модели в этом примере, но может оказаться весьма неэффективной даже для немного более сложных моделей машинного обучения. Генерация вредоносных экземпляров для искажающих атак на разнообразные модели машинного обучения требует применения более эффективных алгоритмов. Ниже описаны два наиболее часто используемых метода на основе аналогичной концепции градиентного подъема:

- метод быстрой смены знака градиента (Fast Gradient Sign Method – FGSM)[1]. Метод FGSM вычисляет градиент выходных данных классификатора с учетом изменений в его входных данных. Находя направление искажения, которое вызывает наибольшее изменение результата классификации, можно однообразно искажать весь блок входных данных в целом (т. е. образ, изображение) с помощью небольших смещений в этом направлении. Этот метод весьма эффективен, но обычно требует искажения большего размера для входных данных, чем требуется для неправильной классификации. Для атакуемых изображений (образов) это означает, что появляется случайный шум (помехи), который покрывает все изображение (образ) в целом;
- методика вычисления якобиана карты выпуклостей (Jacobian Saliency Map Approach – JSMA)[2]. Этот метод генерации вредоносных экземпляров использует концепцию карты выпуклостей (saliency map), т. е. карту относительной важности каждого признака во входных данных. Для изображений такая карта определяет величину изменения пиксела в каждой позиции, необходимую для того, чтобы воздействовать на общий итоговый результат

---

[1] *Ian Goodfellow, Jonathon Shlens and Christian Szegedy*. Explaining and Harnessing Adversarial Examples. ICLR 2015 conference paper (2015).

[2] *Nicolas Papernot et al*. The Limitations of Deep Learning in Adversarial Settings. Proceedings of the 1st IEEE European Symposium on Security and Privacy (2016): 372–387.

классификации. Можно воспользоваться картой выпуклостей для идентификации набора пикселов с наибольшим воздействием, а затем применить метод градиентного подъема для итеративного изменения минимально возможного количества пикселов для приведения к неправильной классификации. Этот метод требует более интенсивных вычислений, чем FGSM, но позволяет создавать вредоносные экземпляры, которые с меньшей вероятностью будут немедленно идентифицироваться людьми-наблюдателями как содержащие искажения.

Как уже отмечалось ранее в данной главе, атаки этого типа могут применяться к разнообразным системам машинного обучения, даже если атакующие обладают чрезвычайно ограниченными знаниями о системе. Другими словами, это атаки типа «черный ящик».

Защита от искажающих атак

На момент написания данной книги не существовало надежных способов защиты от вредоносного искажения данных. Проведенные к настоящему времени исследования показали: все средства, которые проектировщики систем могут применить для защиты от этого класса атак, атакующие могут преодолеть при наличии достаточного времени и вычислительных ресурсов.

Поскольку искажающие атаки основаны на концепции градиентного подъема для поиска экземпляров, принадлежащих к враждебному пространству, общая основополагающая идея защиты моделей машинного обучения против искажающих атак состоит в том, чтобы затруднить для противника получение информации о градиентах поверхности решений модели. Ниже описаны два предлагаемых метода защиты:

- состязательная тренировка (adversarial training). Если модель машинного обучения тренируется на вредоносных экземплярах, то появляется возможность минимизировать враждебное пространство, доступное для использования атакующими. Этот метод защиты пытается перечислить все возможные варианты входных данных для классификатора с помощью выборок данных, принадлежащих к теоретическому пространству входных данных, которое не покрывается исходным распределением тренировочных данных (как показано на рис. 8.1). Тренируемые таким методом модели не должны поддаваться обману на известных им вредоносных экземплярах, но сможем ли мы победить атакующего противника в его собственной игре?

  Состязательная тренировка продемонстрировала многообещающие результаты, но лишь частично решила задачу, так как полный успех этой методики защиты основан на победе в гонке вооружений между атакующими и защищающимися. Поэтому для большинства имеющих смысл проблемных пространств невозможно сформировать исчерпывающее определение всего теоретического пространства входных данных, и атакующий, обладающий достаточной настойчивостью и вычислительными ресурсами, всегда может найти вредоносные экземпляры, которые явно не использовались для тренировки модели;
- защитная дистилляция (defensive distillation). Дистилляция изначально была задумана как методика сокращения (сжатия) размеров моделей ней-

ронных сетей и снижения требований к вычислительным ресурсам, чтобы эти модели могли работать на устройствах с сильно ограниченными ресурсами, таких как мобильные устройства или встроенные системы[1]. Это сокращение (сжатие) достигалось посредством тренировки оптимизированной модели с помощью замены меток категориального класса из исходного тренировочного набора на вектор вероятностей конечных результатов начальной модели. Полученная в результате модель имела более сглаженную поверхность решения, которая в большей степени затрудняла атакующим вывод искомого градиента. Как и состязательная тренировка, метод защитной дистилляции только лишь замедляет и затрудняет для атакующих обнаружение и использование враждебных пространств, следовательно, решает задачу защиты лишь от атакующих с ограниченными вычислительными ресурсами.

Против искажающих атак трудно защищаться исключительно из-за проблемы несовершенного обучения (imperfect learning)[2], т. е. из-за неспособности статистических процессов полностью охватить все возможные входные данные, принадлежащие к конкретной категории объектов, которые позволяли бы выполнять правильную классификацию.

Исследователи в области состязательного машинного обучения разработали CleverHans (**https://github.com/tensorflow/cleverhans**) – библиотеку для эталонного тестирования уязвимости систем машинного обучения к воздействию вредоносных экземпляров. Библиотека предлагает удобный API для применения различных типов атак к произвольно выбираемым моделям, для тренировки локальных заменяющих систем против атак типа «черный ящик» и для проверки эффективности различных методов защиты, таких как состязательная тренировка.

Резюме

Необходимой предпосылкой для создания системы обеспечения безопасности, управляемой методами машинного обучения, является защищенность и надежность самой системы машинного обучения. Несмотря на то что в настоящее время против заражающих и искажающих атак невозможно даже теоретически создать совершенную защиту, это не должно становиться причиной полного отказа от практического использования машинного обучения для обеспечения безопасности. Атаки против систем машинного обучения часто становятся неожиданностью для проектировщиков систем (даже для опытных специалистов-практиков в области машинного обучения), поскольку поведение систем машинного обучения может быть непредсказуемым во враждебных средах. Без полного понимания причин возникновения подобных явлений легко совершить ошибку и неправильно интерпретировать эти результаты как «критические сбои» методов машинного обучения.

---

[1] *Geoffrey Hinton, Oriol Vinyals and Jeff Dean*. Distilling the Knowledge in a Neural Network. Google Inc. (2015).

[2] Эта проблема рассматривалась в одном из предыдущих разделов «Опасные уязвимости в алгоритмах машинного обучения».

Заражающие и искажающие атаки не демонстрируют «критические сбои» методов машинного обучения, а показывают неправильную оценку ожидаемых возможностей машинного обучения на практике. Проектировщики систем машинного обучения должны не удивляться, а обоснованно ожидать, что поведение их систем будет неправильным при неправильном поведении пользователей. Знание типов уязвимостей, которые возникают при использовании систем машинного обучения во враждебных средах, может помочь в качестве стимула к созданию более надежных проектных решений и сокращению количества ложных предположений о возможностях систем машинного обучения.

Приложение A

Дополнительный материал к главе 2

Подробнее о метриках

При обсуждении кластеризации мы пользовались в основном стандартным эвклидовым расстоянием между векторами в пространстве векторов:

$$d(x,y) = \sqrt{\sum_i (x_i - y_i)^2}.$$

Эвклидово расстояние также известно под названием метрики или нормы L_2. Существует также несколько других метрик, которые широко используются в приложениях:

- одним из вариантов эвклидова расстояния является метрика или норма L_1, также называемая расстоянием (метрикой) городских кварталов, или манхэттенской метрикой (потому что она подсчитывает количество «блоков» (кварталов) между двумя точками решетки):

  $$d(x,y) = \sum_i |(x_i - y_i)|;$$

- следующая метрика L_∞, определяемая по следующей формуле:

  $$d(x,y) = \max_i |(x_i - y_i)|;$$

- для векторов бинарных значений, или битов, можно использовать расстояние Хэмминга, т. е. общее количество битов между *x* и *y*. Расстояние Хэмминга можно вычислить следующим образом:

  $$d(x,y) = H(\neg(x \oplus y)),$$

  где $H(v)$ – весовой коэффициент Хэмминга, т. е. число единичных битов в v. Если сравниваемые точки имеют различную длину в битах, то более короткое значение должно быть дополнено префиксом из нулей;
- для списков можно воспользоваться коэффициентом сходства Жаккара:

  $$d(x,y) = \frac{|x \cap y|}{|x \cup y|}.$$

Коэффициент сходства Жаккара вычисляется по количеству элементов, общих для списков *x* и *y*, нормализованному по суммарному количеству элементов в пересечении этих списков. Важным свойством коэффициента сходства Жаккара является возможность его использования для сравнения списков различной длины.

Метрики L1 и L2 в векторных пространствах уязвимы для известной проблемы «проклятия размерности». Это выражение обозначает принцип, согласно которому при увеличении количества измерений все точки выглядят расположенными на приблизительно равном расстоянии друг от друга. Таким образом, если вы пытаетесь выполнить кластеризацию элементов в пространстве с высокой размерностью, то должны либо сократить количество измерений, либо использовать другую метрику, например метрику L_∞. Более формальное описание приведено в разделе 2.5 книги The Elements of Statistical Learning, 2nd ed., by Trevor Hastie, Robert Tibshirani and Jerome Friedman (издательство Springer).

Размер моделей логистической регрессии

Рассматривая более подробно содержимое объекта классификатора `LogisticRegression`, вы заметите, что все изменения после вызова метода `fit()` представляют собой присваивания значений трем атрибутам `coef_`, `intercept_` и `n_iter_`. Исследуем эти атрибуты и постараемся понять, чем в действительности является модель классификатора по методу логистической регрессии:

```
print(clf.coef_)

> [[-7.44949492  0.26692309  1.39595031 -1.44011704  1.41274547
    1.32026309   0.20373255]]

print(clf.intercept_)

> [ 2.93674111]

print(clf.n_iter_)

> [19]
```

Атрибут `n_iter_` не очень важен, поскольку предназначен только для предъявления количества итераций некоторого тренировочного процесса, выполненного для классификатора в его текущем состоянии. Это означает, что полнота (полная сумма) информации, извлекаемой для обучения из тренировочного набора, хранится в восьми числах типа `numpy.float64` атрибутов `coef_` и `intercept_`. Поскольку каждый из восьми объектов типа `numpy.float64` имеет размер 8 байтов, модель может быть полностью представлена с помощью всего лишь 64 байтов при сохранении.

Реализация функции стоимости для метода логистической регрессии

Функция стоимости для бинарной логистической регрессии – это произведение отдельных вероятностей (или правдоподобий – likelihoods) для каждого класса:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m}\left(-y^{(i)}\log(h_\theta(x^{(i)})) - (1-y^{(i)})\log(h_\theta(x^{(i)}))\right).$$

Возможно, эта формула выглядит слегка устрашающе, но в действительности обобщенная концепция не слишком отличается от метода линейной регрессии. Рассмотрим ее подробнее.

В отличие от линейной регрессии, логистическая регрессия – это модель регрессии, в которой зависмая переменная является категориальной, т. е. значение, которое необходимо спрогнозировать, дискретно по своей природе. Это удобно для задач классификации, потому что требуемый итоговый результат будет принадлежать к одной из n категорий меток. Например, классификатор мошеннических платежей, рассмотренный в предыдущем разделе, – это бинарный классификатор, который выводит только результат 0 или 1.

Ошибка (источником которой являются разности) для одной точки далее может быть выражена с использованием логарифмических правдоподобий сигмоидной функции:

$$\mathrm{Err}(h_\theta(x), y) = \begin{cases} -\log(1 - h_\theta(x)), & \text{если } y = 0 \\ -\log h_\theta(x), & \text{если } y = 1 \end{cases}.$$

Это выражение можно переписать следующим образом:

$$\mathrm{Err}(h_\theta(x), y) = -y \log(h_\theta(x)) - (1 - y)\log(1 - h_\theta(x)).$$

Если значение h(x) очень близко к 1, то потеря мала, когда истинная метка 1, и велика, когда истинная метка 0.

Функция стоимости – это просто среднее значение всех ошибок в тренировочном наборе:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \mathrm{Err}(h_\theta(x^{(i)}), y^{(i)}).$$

Если раскрыть это выражение, то получим упомянутую выше функцию стоимости логистической регрессии:

$$J(\theta) = \frac{1}{m}\sum_{i=1}^{m} \left(-y^{(i)} \log(h_\theta(x^{(i)})) - (1 - y^{(i)})\log(1 - h_\theta(x^{(i)}))\right).$$

Минимизация функции стоимости

Интуитивным способом понимания процесса минимизации функции стоимости является рассмотрение более простой формы машинного обучения с учителем – линейной регрессии, также известной под названием оценка регрессии методом наименьших квадратов. При исследовании двумерного набора данных необходимо подогнать линию регрессии (линию наилучшего соответствия, линию тренда) для захвата взаимоотношения между этими двумя измерениями (по осям *x* и *y* на рис. 2.4).

Для этого сначала определим функцию стоимости, которую будем использовать как цель процесса оптимизации. Эта функция стоимости дает нам числовую меру того, насколько успешно линия регрессии способна захватывать линейное взаимоотношение в исследуемых данных. Функция стоимости в соответствии

с определением алгоритма линейной регрессии представляет собой сумму квадратов разностей для каждой точки в наборе данных. Разность для любой точки данных – это разность между прогнозируемыми и действительными значениями *y*, как показано на рис. А.1. Суммирование всех квадратов разностей между точками набора данных и соответствующими координатами линии регрессии дает стоимость этой конкретной линии. Чем больше значение функции стоимости, тем хуже линия регрессии способна захватывать линейное взаимоотношение в исследуемом наборе данных. Таким образом, оптимизация цели – это регулировка (итеративная корректировка) параметров модели линейной регрессии (т. е. угла наклона и точек пересечения линии регрессии) для минимизации выбранной функции стоимости.

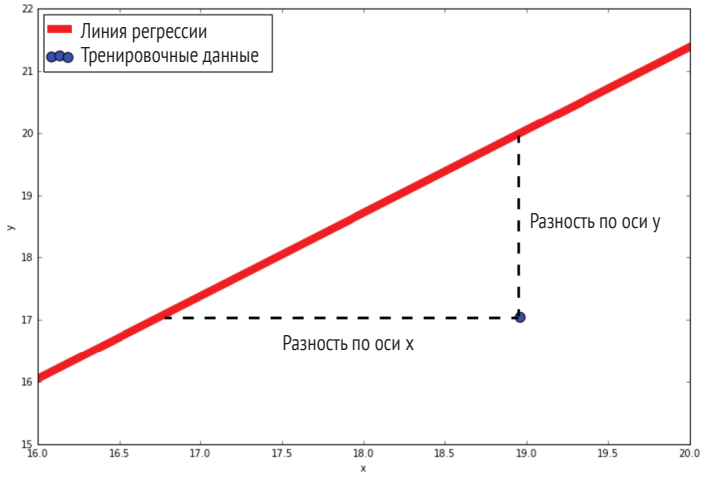


Рис. А.1 ❖ Наглядное определение разностей по осям x и y относительно линии регрессии для одной точки из исследуемого набора данных

Для алгоритмов оптимизации методом градиентного спуска необходимо найти градиент этой функции стоимости, дифференцируя ее по θ:

$$\frac{\partial J(\theta)}{\partial \theta_j} = \frac{1}{m}\sum_{i=1}^{m}\left(h_\theta(x^{(i)}) - y^{(i)}\right)x_j^{(i)}.$$

После того как вы получили некоторое представление о том, как в действительности выполняется процесс тренировки модели регрессии, можно попробовать реализовать собственную версию функции `fit()` из библиотеки scikit-learn. Начнем с определения функций логистики, стоимости и градиента, пользуясь приведенным выше описанием:

```
# Логистическая функция, также известная под названием сигмоидная функция
def logistic(x):
    return 1 / (1 + np.exp(-x))

# Функция стоимости логистической регрессии
def cost(theta, X, y):
```

```
    X = X.values
    y = y.values

    # Отметим, что мы отсекаем минимальные значения, чуть большие нуля,
    # чтобы избежать возникновения ошибки при вычислении логарифма
    log_prob_zero = np.log(
        (1 - logistic(np.dot(X, theta))).clip(min=1e-10))
    log_prob_one = np.log(
        logistic(np.dot(X, theta)).clip(min=1e-10))

    # Вычисление значений логарифма правдоподобия
    zero_likelihood = (1 - y) * log_prob_zero
    one_likelihood = -y * log_prob_one

    # Суммирование по всем экземплярам, затем вычисление среднего значения
    return np.sum(one_likelihood - zero_likelihood) / (len(X))

# Функция градиента логистической регрессии
def gradient(theta, X, y):
    X = X.values
    y = y.values

    num_params = theta.shape[0]
    grad = np.zeros(num_params)
    err = logistic(np.dot(X, theta)) - y

    # Итерация по параметрам и вычисление градиента
    # для каждой исследуемой текущей ошибки
    for i in range(num_params):
        term = np.multiply(err, X[:, i])
        grad[i] = np.sum(term) / len(X)

    return grad
```

Воспользовавшись примером выявления мошеннических платежей из раздела «Машинное обучение на практике: работающий пример» главы 2 (где данные уже считаны и созданы тренировочный и тестовый наборы), продолжим дальнейшую подготовку данных для оптимизации. Отметим, что в данном случае мы пытаемся оптимизировать восемь параметров модели. Наличие k + 1 параметров модели, где k – количество признаков в тренировочном наборе (в рассматриваемом примере k = 7), типично для метода логистической регрессии, так как в нашем распоряжении имеется отдельный «вес» для каждого признака, а кроме того, элемент «bias» (смещение), для того чтобы форма умножаемых матриц была более удобной, вставим в X столбец нулей:

```
# Вставка столбца нулей для более удобного выполнения умножения матриц
X_train.insert(0, 'ones', 1)
X_test.insert(0, 'ones', 1)
```

Далее случайным образом инициализируем параметры модели в массиве с размером 8 и присвоим этому массиву имя `theta`:

```
# Значение seed для воспроизводимости результатов
np.random.seed(17)
theta = np.random.rand(8)
```

В качестве исходного пункта (базовой линии) вычислим стоимость для текущего неоптимизированного состояния модели:

```
cost(theta, X_train, y_train)
```

```
> 20.38085906649756
```

Теперь воспользуемся реализацией алгоритма градиентного спуска, предоставляемой библиотекой SciPy, `scipy.optimize.fmin_tnc` (**https://docs.scipy.org/doc/scipy-0.14.0/reference/generated/scipy.optimize.fmin_tnc.html**). В этой реализации применяется алгоритм оптимизации методом сопряженных градиентов Ньютона[1], оптимизированный вариант простого метода градиентного спуска, описанного выше (в библиотеке scikit-learn можно воспользоваться этим методом решения, определив `solver:'newton-cg'`):

```
from scipy.optimize import fmin_tnc

res = fmin_tnc(func=cost, x0=theta, fprime=gradient,
               args=(X_train, y_train))
```

Результаты оптимизации методом градиентного спуска сохранены в кортеже `res`. Исследуя содержимое `res` (и обращаясь за разъяснениями к документации на примененные функции), можно видеть, что нулевая позиция в этом кортеже содержит решение (т. е. восемь параметров тренируемой модели). Первая позиция содержит количество оценок функции, а вторая – возвращаемый код:

```
> (array([ 19.25533094, -31.22002744,   0.55258124,   4.05403275,
           -3.85452354,  10.60442976,  10.39082921,  12.69257041]), 55, 0)
```

Отсюда понятно, что алгоритм градиентного спуска выполнил 55 итераций для успешного нахождения локального минимума[2] (код возврата 0). Следовательно, параметры модели оптимизированы. Теперь функция стоимости имеет следующее значение:

```
cost(res[0], X_train, y_train)
```

```
> 1.3380705016954436e-07
```

Результат оптимизации выглядит вполне успешным, поскольку стоимость снизилась с начального значения 20,38 до текущего значения 0,0 000 001 338. Теперь проверим оценку этих тренированных параметров на тестовом наборе и понаблюдаем, насколько эффективно тренированная модель логистической регрессии работает в действительности. Сначала определяем функцию `get_predictions()`, которая просто выполняет умножение матриц по тестовым данным, и объект `theta` перед передачей его в логистическую функцию для получения оценки вероятности:

```
def get_predictions(theta, X):
    return [1 if x >= 0.5 else 0 for x in logistic(X.values * theta.T)]
```

---

[1] *R. Fletcher and C. M. Reeves*. Function Minimization by Conjugate Gradients. The Computer Journal 7 (1964): 149–154.

[2] Поскольку эта функция стоимости логистической регрессии является выпуклой, можно не сомневаться в том, что любой найденный локальный минимум является также глобальным минимумом.

Затем выполняем проверку с передачей тестового набора данных и сравнение результатов с тестовыми метками:

```
y_pred_new = get_predictions(np.matrix(res[0]), X_test)
print(accuracy_score(y_pred_new, y_test.values))
```

```
> 1.0
```

На тестовых данных получена 100%-ная точность. Это говорит о том, что оптимизация действительно проведена успешно и мы получили хорошо натренированную модель логистической регрессии.

Приложение Б

Разведка на основе открытых источников

Сообщество профессионалов в области обеспечения безопасности проводит постоянную работу, направленную на защиту периметров, устранение уязвимостей и обезвреживание взломщиков и мошенников. Поскольку злоумышленники, как правило, атакуют одновременно сразу несколько организаций, защищающиеся стороны должны уделить особое внимание совместному использованию и взаимному обмену самой актуальной информацией для укрепления линии обороны. Совместная разведывательная деятельность служб безопасности уже доказала свою исключительную полезность для выявления атак и оценки потенциальных опасностей. Термин «разведка на основе открытых источников» (Open Source Intelligence – OSINT) обозначает сбор данных из разнообразных источников (не обязательно связанных с обеспечением безопасности) и предоставление собранных данных другим системам, которые могут использовать их для выполнения прогнозов и организации ответных действий. Здесь мы кратко рассмотрим некоторые типы разведки на основе открытых источников, а также ее влияние на обеспечение безопасности систем машинного обучения. Наш обзор не претендует на исчерпывающую полноту, для получения более подробной информации по этой теме мы рекомендуем ознакомиться со специальной литературой[1].

Материалы разведки для обеспечения безопасности

Материалы разведки потенциальных угроз могут стать обоюдоострым оружием, когда применяются к системам машинного обучения. Наиболее часто публикуемыми материалами разведки для обеспечения безопасности является список IP-адресов в реальном времени или черный список e-mail. При сборе самых свежих трендов атак и характеристик из ловушек, поисковых роботов, сканеров и проприетарных источников эти материалы предоставляют актуальный список значений, который могут использовать другие системы как признак для классификации

---

[1] *Lee Brotherston and Amanda Berlin*. Defensive Security Handbook: Best Practices for Securing Infrastructure (Sebastopol, CA: O'Reilly Media, 2017), Chapter 18.
Robert Layton and Paul Watters. Automating Open Source Intelligence: Algorithms for OSINT (Waltham, MA: Syngress, 2015).
Sudhanshu Chauhan and Nutan Panda. Hacking Web Intelligence: Open Source Intelligence and Web Reconnaissance Concepts and Techniques (Waltham, MA: Syngress, 2015).

объектов. Например, Spamhaus Project (**https://www.spamhaus.org/**) отслеживает спам, вредоносное ПО и публикует векторы угроз по всему миру, предоставляя в реальном времени материалы по почтовым серверам, взломанным и захваченным серверам и IP-адресам конечных пользователей, для которых собранные данные и аналитики определили явное некорректное поведение в режиме онлайн. Подписчик списков блокировки (blocklists) Spamhaus может запросить конечную точку, чтобы обнаружить источник пришедшего в его систему запроса, который демонстрирует некорректное поведение в интернете. Полученный ответ может стимулировать вторичные решения или действия, такие как увеличение оценки потенциальной опасности входящего запроса, если он был помечен как исходящий с (предположительно) взломанного или захваченного сервера.

Общей проблемой для потребителей материалов разведки потенциальных угроз является надежность и применимость этих материалов в разнообразных системах. То, что определено как угроза в одном контексте, может не являться угрозой в любом другом контексте. Более того, как можно быть совершенно уверенным в том, что предоставляемые материалы надежны и сами по себе не являются средством заражающих атак? Подобные вопросы могут существенно ограничить непосредственное применение материалов разведки потенциальных угроз во многих системах. Threat Intelligence Quotient Test (**https://github.com/mlsecproject/tiq-test**) – это система (в настоящее время не разрабатываемая активно), которая обеспечивает «простое статистическое сравнение различных показателей источников материалов разведки потенциальных угроз, таких как новизна, частичное или полное покрытие, полнота, актуальность («возраст») и уникальность». Подобные инструментальные средства могут помочь количественно оценить и сравнить надежность и полезность материалов о потенциальных угрозах.

Несмотря на присущие им недостатки, материалы разведки потенциальных угроз могут предоставить полезные признаки для пополнения наборов данных или для использования в качестве источника подтверждения, когда система машинного обучения для обеспечения безопасности подозревает, что некоторый объект является вредоносным.

Другой часто применяемый вариант использования материалов разведки потенциальных угроз – обеспечение объектами репутационных систем, которые отслеживают хронологию IP-адреса, домена или хронологию поведения учетной записи пользователя. Организации с большим опытом работы обычно поддерживают сложную комплексную[1] базу знаний об объектах в системе, которая позволяет определять уровень доверия к этим объектам. Например, если некоторый IP-адрес из Восточной Европы постоянно предоставляет материалы разведки потенциальных угроз как хост, потенциально захваченный ботнетом, его оценка в базе данных репутации IP-адресов, вероятнее всего, будет низкой. Когда следующий запрос с этого IP-адреса проявляет даже малейшие признаки аномалии, можно выполнить упреждающее действие по отношению к нему, тогда как IP-

[1] Здесь слово «сложный» (в оригинале – compounding) используется в том же смысле, в каком выражение «сложный процент» (compaund interest) используется в финансовом контексте. Базы знаний часто являются сложными в том смысле, что они используются для создания систем, которые генерируют больше знаний, возвращаемых обратно в исходную базу знаний.

адресу, не имеющему в своем послужном списке отметок о вредоносных действиях, можно оказать большее доверие.

Геолокация

IP-адрес является наиболее частым объектом для идентификации потенциальных угроз для веб-приложений. Поскольку каждый запрос отправляется с какого-либо IP-адреса, а большинство адресов может быть связано с некоторым набором координат физической локации, сбор IP-адресов позволяет аналитикам данных получать информацию об источнике запроса и делать выводы об уровне угрозы. В дополнение к физической локации материалы разведки IP также почти всегда предоставляют номер автономной системы (ASN), данные о провайдере интернет-сервиса (ISP) и даже тип устройства, связанного с IP-адресом. Maxmind (**https://www.maxmind.com/**) – один из самых широко известных провайдеров IP-разведки, который предоставляет часто обновляемые базы данных и API для определения информации о географическом местоположении по любому IP-адресу.

Хотя геолокация является ценным признаком для систем обеспечения безопасности на основе машинного обучения, важно отметить, что существуют некоторые тонкости и хитрости при исследовании IP-адресов, связанных с веб-запросами. Может оказаться, что IP-адрес не соответствует пользователю, отправившему запрос, так как ваша система видит только адрес самого последнего сегмента перехода (last hop) в маршруте запроса. Например, если пользователь размещен за прокси-сервером, то будет виден IP-адрес этого прокси-сервера, а не IP-адрес пользователя. Кроме того, IP-адреса невозможно с уверенностью связывать с конкретным человеком. Многочисленные пользователи в большой семье или в крупной организации могут совместно пользоваться одним IP-адресом, если работают с единственным интернет-соединением или объединены одним и тем же прокси-сервисом. Многие интернет-провайдеры предоставляют динамические IP-адреса, т. е. IP-адреса конечных пользователей постоянно меняются. Пользователи мобильных устройств в сети сотовой связи также обычно имеют периодически меняющиеся IP-адреса, даже если не покидают физическую локацию, потому что на каждом узле сотовой связи существует пул свободных IP-адресов, который совместно используют все подключающиеся пользователи.

Предметный указатель

Б

В

Книги издательства «ДМК Пресс» можно заказать
в торгово-издательском холдинге «Планета Альянс» наложенным платежом,
выслав открытку или письмо по почтовому адресу:
115487, г. Москва, 2-й Нагатинский пр-д, д. 6А.
При оформлении заказа следует указать адрес (полностью),
по которому должны быть высланы книги;
фамилию, имя и отчество получателя.
Желательно также указать свой телефон и электронный адрес.
Эти книги вы можете заказать и в интернет-магазине: **www.a-planeta.ru**.
Оптовые закупки: тел. **(499) 782-38-89**.
Электронный адрес: **books@alians-kniga.ru**.

Кларенс Чио, Дэвид Фримэн

Машинное обучение и безопасность

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод *Снастин А. В.*
Корректор *Синяева Г. И.*
Верстка *Чаннова А. А.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «PT Serif». Печать офсетная.
Усл. печ. л. 31,53. Тираж 200 экз.

Веб-сайт издательства: **www.dmkpress.com**