Рекомендательные системы на практике

Ким Фальк

MANNING

ДМК

Ким Фальк

Рекомендательные системы на практике

Practical Recommender Systems

Kim Falk

MANNING
SHELTER ISLAND

Рекомендательные системы на практике

Ким Фальк

ДМК Пресс
издательство
Москва, 2020

УДК 004.594
ББК 32.972
Ф19

Ким Фальк

Ф19 Рекомендательные системы на практике / пер. с англ. Д. М. Павлова. – М.: ДМК Пресс, 2020. – 448 с.: ил.

ISBN 978-5-97060-774-9

Книга посвящена рекомендательным системам, которые собирают данные о пользователе и выводят для него персональные рекомендации, основываясь на его предпочтениях. Ким Фальк, специалист по обработке и анализу данных, предоставляет читателю самые важные сведения о рекомендательных системах – начиная с общего обзора и описания ключевых алгоритмов до рассмотрения тонких нюансов работы, благодаря которым система с максимальной точностью учитывает интересы пользователя. Помимо прочего, обсуждаются методы оценки рекомендательной системы вне интернета и возможности совмещения различных рекомендательных систем.

Книга снабжена многочисленными примерами программного кода.

Издание предназначено для широкого круга разработчиков и специалистов по анализу данных.

УДК **004.594**
ББК **32.972**

ISBN 978-1-61729-2705 (англ.)
ISBN 978-5-97060-774-9 (рус.)

Посвящается любви всей моей жизни:
моей жене Саре и моему сыну Питеру,
маленькому супергерою

Оглавление

Предисловие от издательства

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**.

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. Издательство «ДМК Пресс» очень серьезно относится 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.

Предисловие

Когда в 2003 году я окончил университет, были опасения, что в Европе программисты будут не востребованы, поскольку все разработки будут проводиться в странах с гораздо более низким уровнем зарплат. К счастью, по множеству причин эти опасения не сбылись. Как я предполагаю, одной из главных проблем было то, что компании недооценивали следующую проблему: разработчики не понимали культурных реалий тех мест, где будут применяться их разработки. Разрабатывалось необходимое программное обеспечение, но функциональные возможности не соответствовали ожиданиям.

В настоящее время аналогичная проблема возникла в сфере машинного обучения и анализа данных. Разница лишь в том, что у ее истоков не низкие зарплаты, а программное обеспечение как услуга (SaaS), система, в которую вы загружаете данные и которая делает за вас всю остальную работу.

Меня, как и всех остальных, беспокоит тот факт, что машины не понимают домены и людей. Машины еще не настолько умны, чтобы можно было исключить человека из задачи. Развитие идет быстрыми темпами, но я рискну предположить, что любой, кто читает эту книгу, сможет работать с рекомендательными системами вплоть до завершения своей трудовой деятельности.

Как я попал в эту сферу? Я работал разработчиком ПО в Италии и собирался переезжать в Англию, где мне хотелось получить работу поинтереснее простого управления данными в базе. К счастью, на меня вышел замечательный сотрудник кадрового агентства RedRock Consulting Ltd. Агентство связало меня с компанией, занимающейся разработкой рекомендательных систем, куда меня взяли для работы над базовым программным обеспечением. И все – я с головой ушел в машинное обучение (мне было невероятно интересно). Помимо работы над рекомендательными системами, я начал изучать информацию об интернете и читал множество книг по этой и схожим тематикам.

Сейчас и чихнуть нельзя без того, чтобы как минимум десять человек не попытались поведать вам какие-то знания из области машинного обучения. Мне очень удивительно видеть одностраничные или одночасовые учебные курсы, авторы которых заявляют, что научат вас всему, что нужно знать о машинном обучении. С таким же успехом я могу написать руководство, как стать летчиком-истребителем:

«Взлетайте и управляйте полетом с помощью штурвала. Если вам нужно стрелять, нажмите кнопку. Прежде чем у вас закончится топливо, необходимо совершить посадку…»

Подобное руководство, возможно, станет для вас неплохой отправной точкой – я начинал точно так же. Но не пытайтесь себя обмануть: для понимания машинного обучения необходимы комплексные знания. Добавьте сюда человеческий фактор, который всегда еще больше все усложняет.

Возвращаясь к моей истории: я работал над рекомендательными системами и получал от этого удовольствие, а затем сменил работу. На новой должно-

сти я должен был продолжить работу над рекомендательными системами, но этот проект был отложен. Тогда я стал переживать, что больше мне не представится возможность работать с рекомендательными системами, и именно в этот момент издательство Manning предложило мне написать о них книгу. Конечно же, я с радостью взялся за это задание. Как только контракт был подписан, проект по рекомендательным системам все-таки начался. Работая над этой книгой, я существенно обогатил свои знания, и надеюсь, что и вам она принесет пользу.

Задача этой книги – познакомить вас с рекомендательными системами: не только с алгоритмами, но и со всей экосистемой. Алгоритмы не отличаются особой сложностью, но, чтобы их понимать и применять, необходимо понимать пользователей, для которых система будет предлагать рекомендации. Содержание книги менялось в процессе ее написания, поскольку я пытался включить в нее как можно больше информации. Надеюсь, что, прочитав эту книгу, вы получите все необходимые знания, чтобы начать работать с рекомендательными системами, и у вас появится база для дальнейшего обучения.

Благодарности

Я хочу отметить и поблагодарить две группы людей: тех, кто активно работал над книгой, и тех, кому приходилось терпеть и поддерживать меня в течение последних трех лет, пока я работал, не обращая внимания ни на что вокруг.

Хотя на обложке «Рекомендательных систем на практике» стоит мое имя, эта книга не появилась бы на свет без активного участия замечательных специалистов из издательства Manning. Я хочу отдельно поблагодарить редактора-консультанта Хелен Стергиус (Helen Stergius) за постоянную помощь и содействие. Она и другие сотрудники превратили мой несколько несвязный текст в руководство, обучающее людей создавать рекомендательные системы.

Также я хочу поблагодарить технических редакторов Фуркана Камаси (Furkan Kamaci) и Валентина Креттаза (Valentin Crettaz), а также всех рецензентов, которые нашли время прочитать первые варианты книги и помогли мне отточить текст, в их числе: Адхир Рамджиаван (Adhir Ramjiavan), Александр Мильцев (Alexander Myltsev), Элвин Радж (Alvin Raj), Амит Ламба (Amit Lamba), Эндрю Колльер (Andrew Collier), Фазел Кештkар (Fazel Keshtkar), Джаред Дункан (Jared Duncan), Яромир Немец (Jaromir Nemec), Мартин Бир (Martin Beer), Маюр Патил (Mayur Patil), Майк Далримпл (Mike Dalrymple), Норин Дертинджер (Noreen Dertinger), Оливье Дукаттеу (Olivier Ducatteeuw), Питер Хэмптон (Peter Hampton), Симеон Лейзерзон (Simeon Leyzerzon), Серен Линд Кристиансен (Søren Lind Kristiansen), Стивен Парр (Steven Parr), Тобиас Бергер (Tobias Bürger), Тобиас Гетрост (Tobias Getrost) и Випул Гупта (Vipul Gupta).

Работая над книгой, я обращался ко множеству различных библиотек, систем и баз данных, и очень благодарен всем сообществам, которые мне помогали. Также я благодарен сообществам разработчиков ПО с открытым исходным кодом, которые создали ряд инструментов, избавив нас от необходимости делать все с нуля.

И в первую очередь я благодарю свою жену, сына и тещу, остальных родственников, а также близких друзей за поддержку, любовь и, самое главное, терпение. Им было нелегко жить с членом семьи, который все время норовит ускользнуть и засесть за написание книги, пока вся семья переезжает в новый дом и лицезреет, как наши дома в Италии разрушаются до основания в результате землетрясений. Не говоря уже о том, что при этом новоиспеченный писатель взялся одновременно за две новые работы. Спасибо вам, и я обещаю, что хотя бы пару лет не буду начинать новые проекты. Люблю вас всех!

О книге

Испытываете ли вы зависть, когда видите рекомендуемые товары Amazon или когда Netflix попадает в точку с рекомендациями подписчикам? Тогда у вас появился шанс пополнить свой арсенал подобными умениями. Читая эту книгу, вы получите представление о том, что такое рекомендательные системы и каково их практическое применение. Чтобы рекомендательная система работала, необходимо, чтобы одновременно выполнялось несколько процессов. Нужно понимать, как собирать данные о пользователях и как их интерпретировать, а также нужно владеть различными алгоритмами рекомендательных систем, чтобы для каждой ситуации можно было выбрать наиболее подходящий из них. И самое главное, нужно уметь чувствовать, хорошо ли функционирует рекомендательная система. Все это и многое другое вы найдете в данной книге.

Кто должен прочитать эту книгу

Книга «Рекомендательные системы на практике» в первую очередь предназначена для разработчиков, которые хотели бы создать рекомендательную систему. В книге собраны практические советы, и материал изложен простым, доступным языком. Есть и вычисления, и статистика, но всегда обязательно присутствуют цифры и программный код. Новоиспеченные специалисты по обработке и анализу данных также почерпнут много полезного из данной книги: получат базовые представления об алгоритмах рекомендательных систем и инфраструктуре, необходимой для их запуска и функционирования. Менеджерам эта книга будет интересна в качестве общего пособия по рекомендательным системам: что это такое и как это можно применить на практике.

Чтобы получить от книги максимум пользы, нужно разбираться в языках программирования, таких как Python и Java, понимать SQL-запросы и обладать базовыми знаниями высшей математики и статистики. Цифры и листинги кодов, которые приводятся для наглядности, дают лишь самое общее представление о теме.

Структура книги

Книга делится на две части: первая посвящена инфраструктуре рекомендательных систем а вторая – алгоритмам.

Из части I вы узнаете, как после добавления рекомендательной системы в приложение получать данные и применять их:

- глава 1 представляет собой общий обзор рекомендательных систем и их ключевых элементов. Она знакомит читателя с рекомендательными системами в целом и базовыми принципами их работы;

- глава 2 говорит о том, как научиться понимать пользователей и их поведение, а также перечисляет способы сбора данных о пользователях;
- глава 3 посвящена веб-аналитике и рассказывает, как можно создать сводную информационную панель для отслеживания данных о своих рекомендательных системах;
- глава 4 говорит о том, как на основе данных о поведении пользователей составлять рейтинги;
- глава 5 рассматривает общие рекомендации (без учета индивидуальных предпочтений пользователя);
- глава 6 описывает проблемы, связанные с новыми пользователями и товарами, и предлагает простые решения.

Часть II рассказывает об алгоритмах рекомендательных систем, а также о том, как на основе собранных системой данных выстроить рекомендации для пользователя:

- глава 7 говорит о формулах для определения степени сходства между различными пользователями или контентом, например фильмами;
- глава 8 рассказывает о составлении персональных рекомендаций путем совместной фильтрации;
- глава 9 приводит методы оценки рекомендательной системы вне интернета и описывает способы составления рекомендаций онлайн;
- глава 10 знакомит вас с фильтрацией по контенту, которая позволяет найти сходные черты в контенте с помощью различных типов алгоритмов, таких как латентное размещение Дирихле и TF-IDF;
- глава 11 возвращается к обсуждению совместной фильтрации, о которой говорилось в главе 8, но сосредотачивается на методах снижения размерности;
- глава 12 рассказывает, как можно совместить различные типы рекомендательных систем;
- глава 13 говорит об алгоритмах ранжирования и способах оценивания рекомендаций;
- глава 14 подводит общий итог, дает задел на будущее, рассказывает, какие вопросы необходимо изучить и какие книги нужно еще прочитать, а также рассуждает об алгоритмах и контексте.

Книга построена таким образом, что лучше ее читать от начала до конца, поскольку во многих местах есть отсылки к предыдущим главам, но вполне допустимо читать и отдельные главы.

Загрузки

Программный код для запуска демонстрационного сайта под названием MovieGEEKs можно загрузить с сайта издательства по ссылке **www.manning.com/books/practical-recommendersystems**, а также с сервиса Github.com по ссылке **mng.bz/04K5**. Сайт создан с помощью фреймворка Django. У нас будет два набора данных: один генерируется автоматически, а второй нужно загрузить из базы MovieTweetings. Все инструкции по установке можно найти на сайте GitHub.

Формат кода

В книге содержится много примеров программного кода – как в виде пронумерованных листингов, так и в составе обычного текста. И в том, и в другом случае исходный код выделен шрифтом определенной ширины, чтобы его было легко отличить от обычного текста. Иногда код также выделен жирным шрифтом, чтобы акцентировать внимание читателя на измененных фрагментах, например при добавлении новой функции в строку кода, приведенную ранее в той же главе.

Во многих случаях мы внесли коррективы в первоначальный формат исходного кода, например добавили переносы строк и изменили отступы, чтобы максимально продуктивно использовать пространство книжной страницы. В редких случаях даже этого было недостаточно, и тогда мы вынуждены были применять символы, указывающие на продолжение строки (➥). Кроме того, из листингов часто приходилось убирать комментарии к исходному коду (когда код описывается в тексте). Многие листинги сопровождаются аннотациями к коду, в которых отражаются важные моменты.

Форум для читателей книги

При покупке «Рекомендательных систем на практике» читатель получает доступ к закрытому интернет-форуму издательства Manning Publications, где можно оставить комментарии о книге, задать технические вопросы и получить помощь от автора и других пользователей. Для входа на форум пройдите по ссылке **forums.manning.com/forums/practical-recommendersystems**. Подробнее познакомиться с форумами издательства Manning и правилами поведения на них можно на странице **forums.manning.com/forums/about**.

Издательство Manning стремится предоставить читателям площадку, где читатели могут общаться друг с другом или с автором книги. Издательство не берет на себя ответственности за обязательное привлечение на форум автора, чье участие в обсуждениях на форуме исключительно добровольное (и неоплачиваемое). Советуем задавать автору интересные вопросы, чтобы вовлечь его в общение! Форум и архивы старых обсуждений будут доступны на сайте издательства, пока книгу не снимут с печати.

Об авторе

Ким Фальк – специалист по обработке и анализу данных, обладающий большим опытом создания приложений, управляемых данными. Рекомендательные системы и машинное обучение в целом вызывают его живейший интерес. Он разрабатывал рекомендательные системы, предлагающие конечным пользователям фильмы и рекламу, и даже помогал юристам находить материалы по судебным прецедентам. Он занимается большими данными и машинным обучением с 2010 года. Ким часто говорит и пишет о рекомендательных системах. Его веб-страница: **kimfalk.org**.

Когда Ким не занят тем, что обучает машины следить за людьми, он отличный муж, отец и владелец курцхаара.

Об обложке

Иллюстрация на обложке «Рекомендательных систем на практике» – это работа под названием «Amazone d'Afrique», т. е. «Африканская амазонка». Этот рисунок Жака Грассе де Сен-Совера (1757–1810) является частью серии изображений национальных костюмов различных стран под названием «Costumes de Différents», опубликованной во Франции в 1797 году. Каждая работа тщательно прорисована и раскрашена вручную.

Богатое разнообразие рисунков в серии Жака Грассе де Сен-Совера напоминает нам о том, насколько сильно разнилась культура городов и регионов мира всего 200 лет назад. Люди, жившие изолированно друг от друга, разговаривали на разных диалектах и языках. На городских улицах или в сельской местности по одежде человека можно было легко определить, чем он занимается и где живет. С тех пор наша манера одеваться сильно изменилась, а региональное разнообразие, столь богатое в те времена, стерлось. Теперь трудно различить жителей разных континентов, а тем более разных стран, регионов и городов. Похоже, что мы пожертвовали культурным многообразием ради того, чтобы достичь большего разнообразия в личной жизни каждого отдельного человека, которая, безусловно, теперь более многогранна, динамична и технологична.

Во времена, когда трудно отличить одну компьютерную книгу от другой, издательство Manning привносит в компьютерную индустрию элемент новизны и свежести, создавая обложки, воспроизводящие богатое региональное разнообразие, которое было присуще миру два века назад и запечатлено на рисунках Жака Грассе де Сен-Совера.

ЧАСТЬ I

Подготовка к рекомендательным системам

Окружающая среда – это все, что не является мной.
Альберт Эйнштейн

Применение рекомендательных систем и, по сути, большинства методов машинного обучения в условиях промышленной эксплуатации подразумевает не только применения наиболее подходящего алгоритма, но и требует понимания пользователей и сферы деятельности.

Главы 1–6, т. е. часть I «Рекомендательных систем на практике», познакомят вас с экосистемой и инфраструктурой рекомендательных систем. Вы научитесь собирать данные и применять их, добавив рекомендательную систему в приложение. Вы узнаете, чем отличается рекомендация от рекламы и персональная рекомендация от неперсональной. Вы также узнаете, как собирать данные для создания собственной рекомендательной системы.

Глава 1

Что такое рекомендательная система?

Чтобы понять, что такое рекомендательная система, необходимо разобраться в огромном объеме информации, поэтому мы начнем эту книгу со следующих вопросов: какие проблемы решает рекомендательная система и как она применяется. **Вот о чем мы будем говорить:**

- выясним, какую задачу пытается выполнить рекомендательная система;
- разберемся, что такое персональные и неперсональные рекомендации;
- разработаем терминологию, позволяющую описывать рекомендательные системы;
- обсудим сайт-образец MovieGEEKs.

Налейте себе чашечку кофе, закутайтесь в одеяло и устройтесь поудобнее – сейчас вы познакомитесь с миром рекомендательных систем. Мы постараемся упростить задачу и сначала рассмотрим примеры из жизни, а только потом, в следующих главах, перейдем к математическим тонкостям рекомендательных систем. Возможно, вам захочется пропустить первую главу и читать дальше, но не следует этого делать. Чтобы получить представление о том, как должны выглядеть результаты вашей работы над рекомендательной системой, необходимо начать с самых азов.

1.1. Рекомендации в реальной жизни

Я много лет жил в Италии, в Риме. Рим – это красивый город, где множество продуктовых рынков – не тех, которые в центре и о которых рассказывается в путеводителях, где среди прочего торгуют поддельными сумками Gucci (да, Gucci на продуктовом рынке), а тех, которые остались за рамками маршрутов туристических автобусов, где закупаются местные жители и торгуют фермеры.

Каждое воскресенье мы покупали овощи и фрукты у продавца по имени Марино. Мы были хорошими покупателями, настоящими гурманами, поэтому он знал, что, если посоветует нам какие-то качественные продукты, мы их купим, хотя изначально собирались закупаться строго по списку. Восхитительные сезонные арбузы, множество разновидностей помидоров, даривших нам настоящий фейерверк вкусов, и потрясающая свежая моцарелла, которую я никогда не смогу забыть. Иногда Марино не советовал покупать нам продукты, качество которых было не на высоте, и мы доверяли его мнению. Это пример рекомендаций. Марино постоянно советовал нам одно и то же, что нормально, когда дело касается еды, но не вариант, когда речь идет о других вещах – например, книгах, фильмах или музыке.

Когда я был моложе, еще до того, как стриминговые сервисы типа Spotify захватили музыкальный рынок, мне нравилось покупать компакт-диски. Я шел в музыкальный магазин, который был ориентирован в первую очередь на диджеев, набирал огромную кипу компакт-дисков, находил у прилавка свободные наушники и начинал слушать. Я подолгу разговаривал с человеком за прилавком по поводу этих компакт-дисков. Он смотрел, какие диски мне понравились (и не понравились) и, исходя из этого, советовал другие. Я очень ценил то, что он запоминал мои предпочтения и не советовал мне одно и то же по нескольку раз. Это тоже пример рекомендаций.

Возвращаясь домой с работы (теперь, когда я старше), я всегда заглядываю в почтовый ящик, проверяя, нет ли писем. Как правило, ящик забит рекламой из супермаркетов, где перечислены товары по акции. Обычно в этих брошюрах на одной странице изображены свежие фрукты, а на другой – средство для посудомоечных машин, т. е. то, что супермаркеты рекомендуют вам купить, подчеркивая, что это выгодно. Это не рекомендации, а *реклама*.

Раз в неделю в почтовом ящике появляется местная газета. В газете публикуют список десяти самых популярных фильмов, которые показывают в кино на этой неделе. Это *неперсональные рекомендации*. На ТВ много внимания уделяется тому, чтобы вставить рекламный блок в подходящий материал. Это *таргетированная реклама*, поскольку считается, что ее смотрят люди определенного склада.

В феврале 2015 года сотрудники копенгагенского аэропорта заявили, что на территории аэропорта было установлено 600 мониторов, на которых, наряду с информацией о рейсе и воротах, транслируется реклама, подборка которой зависит от предполагаемого возраста и пола человека. Данные о возрасте и поле строились на основе полученного с камер изображения и специального алгоритма. В пресс-релизе об этом нововведении говорилось следующее: «Женщине, летящей в Брюссель, будет интересна реклама хороших часов или, например, финансового журнала. Семья, отправившаяся в отпуск, возможно, заинтересуется рекламой солнцезащитного крема или сервиса аренды машин»[1]. Это *релевантная реклама*, т. е. *максимально таргетированная*.

[1] Подробнее об этом читайте по ссылке **mng.bz/ka6j**.

Реклама по телевизору или в аэропорту обычно раздражает людей, но в сети границы того, что мы считаем назойливостью, несколько иные. Тому есть ряд причин, это само по себе является отдельной темой для обсуждения.

Интернет – это по-прежнему Дикий Запад, и, хотя я считаю, что реклама в аэропорту Копенгагена достаточно навязчива, меня не меньше раздражает реклама в интернете, если она предназначена для целевой группы, к которой я не отношусь. Чтобы показывать рекламу определенной целевой группе, веб-сайты должны немного представлять, кто вы такой.

В этой и следующих главах вы узнаете, что такое рекомендации, как собирать информацию о тех, кто будет эти рекомендации получать, как хранить данные и как их применять. Составлять рекомендации можно разными способами, и вы познакомитесь с наиболее распространенными приемами.

Рекомендательная система – это не только хитросплетенный алгоритм. В ее основе также лежит понимание данных и пользователей. Специалисты по анализу данных давно спорят о том, что важнее: наличие суперумного алгоритма или большого объема данных. Везде есть свои сложности. Суперумный алгоритм требует много супертехнологичного оборудования. Большой объем данных влечет за собой другие трудности, например как обеспечить быстрый доступ к этим данным. Читая эту книгу, вы узнаете, где можно пойти на компромисс, и научитесь принимать удачные решения.

Вышеописанные примеры призваны показать, что пользователь может не видеть разницы между рекламой и рекомендациями. Но на самом деле разница в предназначении: *рекомендация* выдается на основе: вкусов пользователя (если он достаточно активен), вкусов других пользователей и того, что чаще всего ищет этот человек. Реклама работает только на благо рекламодателя и обычно навязывается получателю. Различие может быть очень размытым. В этой книге я называю рекомендацией все то, что строится на основании полученных данных.

1.1.1. Рекомендательные системы дома в интернете

Рекомендательные системы чаще всего предполагают домашнее использование через интернет, поскольку так можно не только охватить отдельных пользователей, но и получить данные об их поведении. **Давайте рассмотрим несколько примеров.**

Сайт со списком 10 самых популярных хлебопечек предоставляет *неперсональные* рекомендации. Если сайт с товарами для дома или с билетами на концерты предлагает вам рекомендации, ориентируясь на демографические данные или на ваше текущее местоположение, это *полуперсональные* рекомендации. Персональные рекомендации можно увидеть, например, на сайте Amazon, где зарегистрированные пользователи видят рекомендации в специальном разделе. Потребность в персональных рекомендациях связана с тем, что людям интересны не только популярные товары, но и те, которые не входят в число наиболее продаваемых, а также те, которые находятся в длинном хвосте.

1.1.2. Длинный хвост

Понятие *длинного хвоста* ввел Крис Андерсон в своей статье для журнала Wired в 2004 году, а в 2006 году на основе этой статьи была написана книга[1]. В статье Андерсон описал новую бизнес-модель, которая часто встречается в интернете. Основная идея Андерсона заключалась в том, что, если у вас магазин не в интернете, у вас ограниченное пространство для хранения и, что еще важнее, ограниченное пространство для демонстрации товаров покупателям. Также у вас небольшой круг покупателей, поскольку людям приходится идти в магазин. Если бы не было этих ограничений, вам не пришлось бы продавать только популярные товары, как часто бывает в традиционной торговой бизнес-модели. В офлайн-магазинах держать на складе непопулярные товары считается проигрышной стратегией, поскольку необходимо место для хранения большого количества товаров, которые, возможно, никто не купит. Но если у вас интернет-магазин, в вашем распоряжении место для бесконечного количества товаров, поскольку плата за него невысока, или, если вы продаете цифровой контент, складское пространство вам вообще не требуется, т. е. плата минимальна или равна нулю. Суть экономики «длинного хвоста» сводится к тому, что можно получать доход, продавая много разных товаров, каждый из них – лишь по нескольку экземпляров множеству разных людей.

Я всеми руками за разнообразие, поэтому мне кажется, что огромный каталог товаров – это прекрасно, но вопрос, на который трудно дать ответ, заключается в том, как именно пользователи находят то, что им нужно? Именно тут на сцену выходят рекомендательные системы. Именно такие системы помогают людям находить различные вещи, о существовании которых те даже не догадывались.

В сети титанами в отношении контента и рекомендаций считаются сервисы Amazon и Netflix, поэтому именно они фигурируют в различных примерах в данной книге. В следующем разделе мы рассмотрим сервис Netflix как наглядный пример рекомендательной системы.

1.1.3. Рекомендательная система Netflix

Как вы, вероятно, знаете, Netflix – это стриминговый сервис. Его основной контент – это фильмы и сериалы, подборка которых постоянно обновляется. Задача, которую выполняют рекомендации Netflix, состоит в том, чтобы поддерживать ваш интерес к представленному контенту как можно дольше, стимулируя вас оплачивать подписку из месяца в месяц.

Сервис работает на различных платформах, поэтому рекомендации могут выглядеть по-разному. На рис. 1.1 показан снимок экрана моего ноутбука с открытым сервисом Netflix. Также я могу запустить Netflix на телевизоре, планшете и даже телефоне. На разных устройствах хочется смотреть разное – я никогда не смотрю эпичные фильмы жанра фэнтези на телефоне, но мне нравится смотреть их на телевизоре.

---

[1] Более подробную информацию см. на странице **www.wired.com/2004/10/tail/**. Информацию о книге см. на странице **en.wikipedia.org/wiki/The_Long_Tail_(book)**.

Рис. 1.1. Главная страница сервиса Netflix (до того, как сменился макет)

Начнем нашу ознакомительную экскурсию с главной страницы. Стартовая страница представляет собой панель с названиями категорий, например **Top Picks** (Самое интересное), **Drama** (Драмы) и **Popular on Netflix** (Популярное на Netflix). Верхняя категория – это мои просмотры. Netflix уделяет большое внимание этой категории, поскольку она показывает не только то, что я посмотрел и что смотрю сейчас, но и то, что меня (хотя бы в какой-то степени) заинтересовало.

Netflix стремится обратить ваше внимание на категорию, расположенную строчкой ниже, – **Netflix Originals**, – поскольку в нее входят сериалы производства Netflix. Они важны для Netflix по двум причинам, обе из которых связаны с финансами:

- Netflix тратит огромные деньги на производство собственного контента, который преимущественно можно посмотреть только через сервис Netflix;

- Netflix платит деньги владельцам контента, когда пользователи смотрят этот контент. Если владелец сам Netflix, сервис не просто экономит, а зарабатывает.

Последний пункт – это отдельная пища для размышлений: даже если на странице все персонализировано, категория Netflix Originals все равно идет второй строкой, и сам этот факт, как правило, не свидетельствует о том, что я смотрю этот контент, а демонстрирует стоящие перед компанией бизнес-задачи.

Подборки и тренды

Ниже расположена категория **Trending Now** (Сейчас в тренде). Тренд – это достаточно размытое понятие, под которым многое может иметься в виду, но в данном случае речь идет о контенте, который пользовался популярностью в самое недавнее время. Далее следует категория **Popular on Netflix** (Популярное на Netflix), которая также включает в себя популярный контент, но период времени, в течение которого этот контент пользуется популярностью, дольше – например, около недели. Мы подробно поговорим о трендах и подборках в главе 5.

Рекомендации

Четвертая категория – это персональная подборка **Top Picks** (Лучшее), которая отражает мои предпочтения. Сюда входит все то, что большинство людей назвали бы рекомендациями. Здесь показан контент, который, по прогнозам рекомендательной системы Netflix, я захочу посмотреть в ближайшее время. Часто эти прогнозы верны. Я не фанат кровавых фильмов со сценами жестокости и предпочту не видеть физического насилия ни в каком варианте. Не все предложенные варианты мне по вкусу, но я думаю, что Netflix составляет рекомендации на основе не только моих предпочтений. Другие члены семьи тоже иногда смотрят фильмы и сериалы через мой профиль. *Профили* позволяют сервису Netflix идентифицировать человека, который пользуется данной учетной записью в настоящий момент.

До появления профилей сервис Netflix составлял рекомендации для всей семьи, а не для отдельных пользователей[1]. Он пытался всегда подобрать что-то для мамы, папы и детей. Однако теперь Netflix больше так не делает, поэтому в моем списке отсутствуют детские передачи. Но, даже несмотря на то что сейчас сервис Netflix позволяет создать персональный профиль для каждого пользователя, я считаю, что система обязательно должна принимать во внимание всех зрителей – не только того человека, которому принадлежит профиль, но и остальных. До меня дошли слухи, что другие компании пытаются разработать решение, с помощью которого можно будет проинформировать систему о наличии других зрителей, помимо вас. Так сервис получит возможность предлагать рекомендации, ориентированные на всех присутствующих. На практике пока что я такого ни разу не встречал.

Технология Kinect компании Microsoft была способна идентифицировать сидящих перед телевизором людей с помощью функции распознавания лиц и движений. Разработчики Microsoft пошли еще дальше, и система могла уз-

[1] Netflix Recommendations: Beyond the 5 stars (Part 1), **mng.bz/bG2x**.

навать не только членов семьи, но и других людей из полного каталога пользователей, т. е. она идентифицировала пользователей, находящихся в гостях у других пользователей. Несмотря на этот шаг в сторону распознавания аудитории, сенсор Kinect для Xbox One был снят с производства в октябре 2017 года, что ознаменовало закрытие линейки Kinect.

Категории и разделы

Вернемся к категории **Top Picks** (Лучшее) на Netflix. Если навести курсор на один из предложенных фильмов или сериалов, появится более подробная информация о контенте. Наряду с всплывающим описанием (рис. 1.2) отобразится предполагаемая оценка, которую я, по прогнозу рекомендательной системы, скорее всего, дам этому контенту. Логично предположить, что все видео из категории **Top Picks** будут с высокими оценками, как на рис. 1.1. Однако, просматривая рекомендации, можно встретить варианты с прогнозом на низкие оценки, как на рис. 1.3.

Netflix формирует рекомендации множеством разных способов, и можно найти множество возможных объяснений тому, почему Netflix рекомендует контент, который, по собственным же прогнозам сервиса, получит от пользователя невысокую оценку. Одна из причин, возможно, заключается в том, что Netflix ставит разнообразие превыше точности попадания. Другая вероятная причина заключается в том, что, пусть я и не поставлю фильму максимальное количество звезд, вдруг именно этот фильм сейчас подходит под мое настроение. Кроме того, это первый признак, что Netflix не придает этим оценкам большого значения.

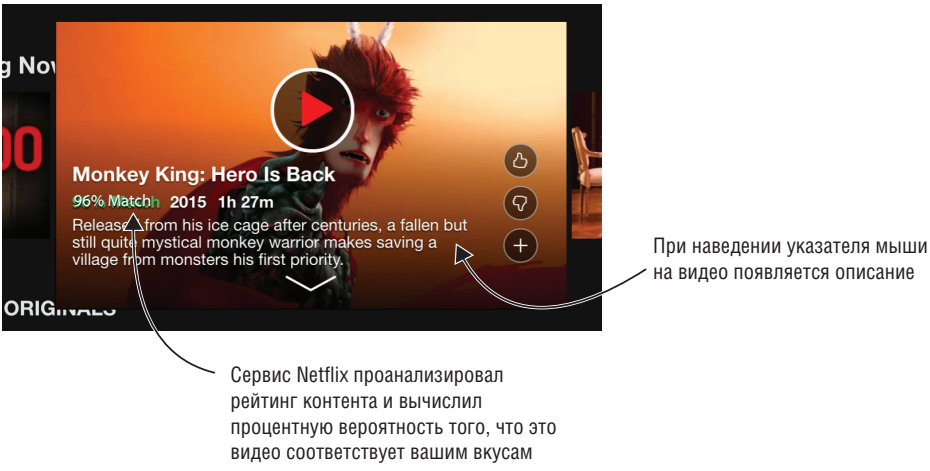


Рис. 1.2. Мультфильм из категории Top Picks на Netflix и прогнозируемое соответствие вкусам пользователя

У всех разделов разные названия. Некоторые формируются по принципу «Потому что вы посмотрели “Форс-мажоры”». Туда включены фильмы и сериалы, похожие на «Форс-мажоры». Другие разделы озаглавлены в зависимости от представленных жанров: например, раздел **Comedies** (Комедии), как ни удивительно,

включает в себя комедии. В принципе, названия разделов – это тоже в какой-то степени рекомендации, т. е. эти *категории* также можно считать *рекомендациями*.

Тут можно было бы и остановиться, но тогда мы бы пропустили самый важный аспект персонализации на Netflix.

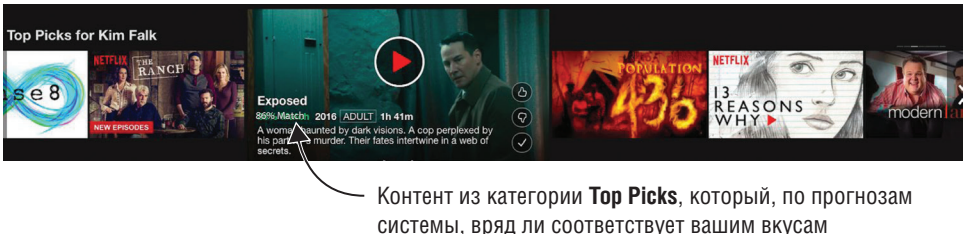


Рис. 1.3. Фильм из категории Top Picks на Netflix с прогнозируемо низкой оценкой.

Оценки

Заголовок каждой категории описывает данную подборку контента. Видео в подборке расположено в соответствии с алгоритмом рекомендательной системы – в порядке релевантности или по оценке, слева направо, как показано на рис. 1.4.

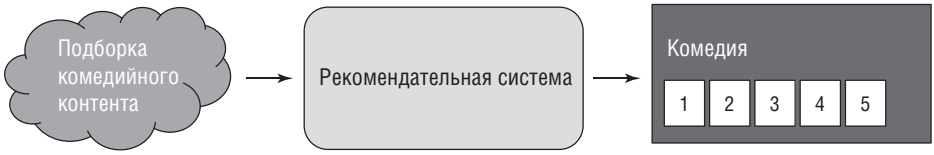


Рис. 1.4. На Netflix контент в каждой строке расположен по релевантности.

Даже в категории **My List** (Мой список), куда входит тот контент, который я выбрал самостоятельно, видео располагается в порядке, предусмотренном рекомендательной системой, – в соответствии с предполагаемой релевантностью моим предпочтениям. На рис. 1.1 представлен вчерашний снимок экрана. Сегодня видео в моем списке уже расположено по-другому, как показано на рис. 1.5.

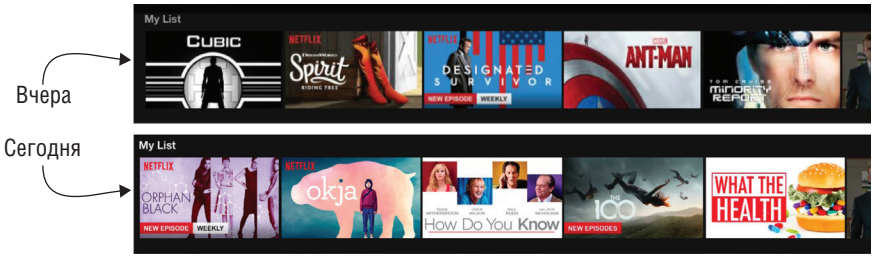


Рис. 1.5. Netflix выстраивает видео в моем списке в порядке релевантности.

Рекомендательная система Netflix также пытается предлагать контент, который будет актуален в определенное время или при определенных условиях. На-

пример, воскресное утро больше подходит для просмотра мультфильмов и комедий, а вечер – для просмотра «серьезных» сериалов, таких как «Форс-мажоры».

Еще одна категория, содержимое которой может вас удивить, это – **Popular on Netflix** (Популярное на Netflix), куда входит контент, пользующийся популярностью в данный момент. Однако крайнее слева видео в этой категории вовсе не обязательно самое популярное. Netflix находит несколько самых популярных вещей, а затем располагает их в таком порядке, который, с точки зрения рекомендательной системы, в наибольшей степени отвечает вашим предпочтениям.

Продвижение

Любопытный вопрос – почему Netflix поставил на одно из первых мест в категории **My List** сериал «Последний кандидат», учитывая, что я уже и так его смотрю. Netflix добавил пометку, что вышел новый сезон «Последнего кандидата». Возможно, это и есть причина появления этого сериала в категории **My List**.

Продвижение – это один из способов склонить чашу весов в ту или иную сторону при составлении рекомендаций. Например, Netflix хочет, чтобы я обратил внимание на сериал «Форс-мажоры», потому что это новый контент, а значит, его ценность выше. Netflix продвигает новый контент. Под *новизной* можно понимать, что этот контент только появился или мелькнул в новостях. В главе 6 мы подробно поговорим о продвижении, поскольку именно продвижение начинает интересовать многих владельцев сайтов, как только система отлажена и запущена.

> **ПРИМЕЧАНИЕ.** Существует еще понятие бустинга (продвижения), относящееся к алгоритмам машинного обучения. Но я имею в виду не его[1].

Синхронизация с социальными сетями

На протяжении короткого периода времени сервис Netflix также пытался применять данные из социальных сетей[2]. Тогда на главной странице Netflix можно было увидеть что-то вроде того, что показано на рис. 1.6.

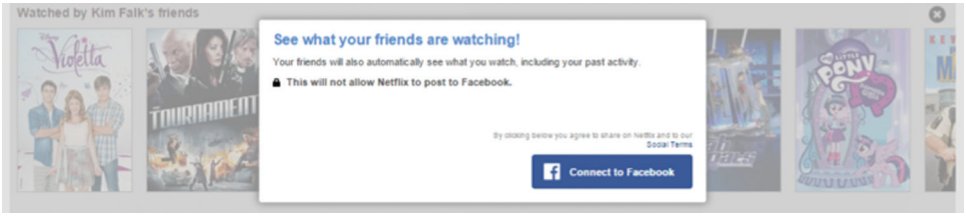


Рис. 1.6. Netflix хочет знать, что смотрят мои друзья

Netflix предлагал привязать к учетной записи свой аккаунт на Facebook, предоставив себе доступ к списку ваших друзей, а также к другой информации. Одним из преимуществ этой привязки для Netflix было то, что так систе-

[1] См. en.wikipedia.org/wiki/Boosting_(machine_learning).

[2] См. **mng.bz/6yHM.**

ма могла изучать ваших друзей и составлять общие рекомендации, опираясь на их предпочтения. Привязка к Facebook также могла превратить просмотр фильмов в совместное мероприятие – очень популярное направление, которое развивают многие медиакомпании.

В наши дни люди не сидят и не смотрят фильмы просто так. Они многозадачны: смотрят кино и одновременно зависают в другом устройстве (например, планшете или смартфоне). От того, чем вы занимаетесь на втором устройстве, может во многом зависеть, что вы будете смотреть дальше. Представьте себе, что, посмотрев что-то на Netflix, вы видите на телефоне уведомление о том, что кому-то из ваших друзей понравился какой-нибудь фильм, и вуаля – Netflix рекомендует вам посмотреть этот фильм следующим.

И все-таки этот функционал был отключен году в 2015-м или 2016-м по той причине, что люди не хотели делиться информацией о своих фильмах с контактами на Facebook. Как сказал Нил Хант, директор по контенту Netflix: «Это очень печально, поскольку я считаю, что, совместив рекомендации по алгоритму и персональные рекомендации, можно получить ценный ресурс»[1].

Профиль предпочтений

Если страница практически полностью строится на основе ваших предпочтений, неплохо бы предоставить системе как можно больше информации о своих вкусах. Если у Netflix не будет ясного понимания, что вам нравится, вероятно, вам будет трудно найти что-нибудь интересное для себя.

В 2016 году сервис Netflix предоставлял пользователям возможность настроить профиль. Меню **Taste Profile** (Профиль предпочтений), показанное на рис. 1.7, позволяло оценивать сериалы и фильмы, выбирать жанры, указывая, как часто вам хочется посмотреть что-нибудь остросюжетное (Netflix называет этот тип фильмов **Adrenaline Rush** (Адреналиновая лихорадка), см. рис. 1.8) или проверить оценки, которые вы выставили ранее, и убедиться, что ваше мнение не изменилось.

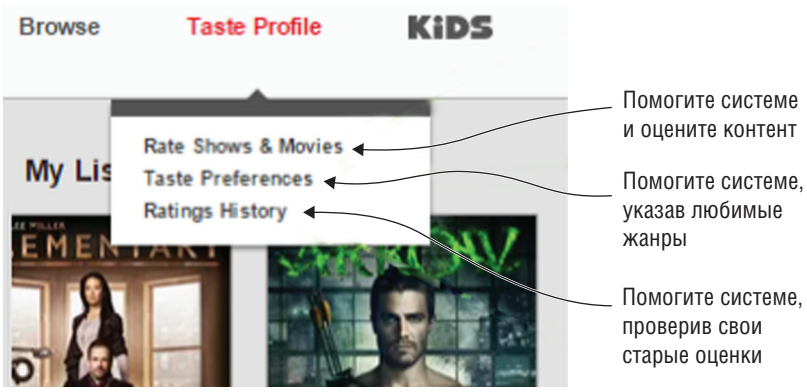


Рис. 1.7. Пример того, как выглядели профили предпочтений Netflix в 2015 году

[1] См. **mng.bz/jc7M**.

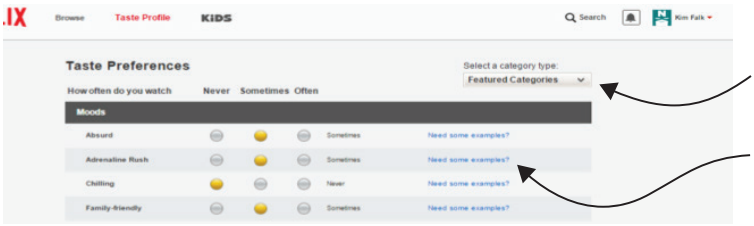


Рис. 1.8. Меню профиля предпочтений Netflix

Благодаря этим настройкам рекомендации Netflix стали точнее. Обратиться к пользователю за помощью при составлении профиля предпочтений – это достаточно распространенный прием, к которому прибегают системы при формировании рекомендаций для новых пользователей. Но, как часто бывает, между тем, что пользователям нравится, по их словам, и тем, что им нравится на самом деле, существует большая разница.

Профиль предпочтений – это первая ступенька на пути к знакомству с пользователем. Чем дольше пользователь взаимодействует с системой, тем больше данных о нем собирает Netflix, и, как правило, на эти данные можно полагаться целиком и полностью. К настоящему моменту Netflix отказался от заполняемого пользователями профиля предпочтений.

1.1.4. Определение рекомендательной системы

Чтобы убедиться, что мы правильно понимаем друг друга, посмотрите табл. 1.1, где указаны определения основных понятий.

Таблица 1.1. Рекомендательные системы: термины и понятия

Термин	Пример из Netflix	Определение
Прогноз	Netflix угадывает, какую оценку вы поставите контенту	Прогноз – это предположение относительно того, насколько пользователю понравится контент
Релевантность	Расположение категорий контента на странице (например, **Top Picks** (Лучшее) и **Popular on Netflix** (Популярное на Netflix)) по степени интересности	Расположение контента в соответствии с тем, что больше всего подходит пользователю в данный момент. Релевантность сочетает в себе контекст, демографические данные и (ожидаемые) оценки
Рекомендация	**Top Picks** (Лучшее) для меня	Лидеры по релевантности
Персонализация	Заголовки категорий на странице – это пример персонализации	Сочетание релевантности и наглядности
Профиль предпочтений	См. рис. 1.8	Список характеристик и их значений

Разобравшись с этими терминами, мы, наконец, можем дать определение рекомендательной системе.

Определение: рекомендательная система

Рекомендательная система подбирает и предлагает пользователю релевантный контент, основываясь на своих знаниях о пользователе, контенте и взаимодействии пользователя и контента.

Прочитав это определение, вы, возможно, решили, что вам все понятно. Но давайте рассмотрим пример того, как подбираются рекомендации и как работает система. На рис. 1.9 показано, каким образом сервис Netflix мог сформировать для меня подборку **Top Picks** (Лучшее). Вот как мог выглядеть процесс подбора контента в данном случае (по шагам):

1. Получен запрос на подборку **Top Picks** (Лучшее).
2. Сервер обращается к рекомендательной системе, в этом участвует целый ряд методов. Этот шаг называется *получить варианты объектов*. С его помощью сервер из базы данных каталога получает объекты, которые в наибольшей степени релевантны предпочтениям пользователя в данный момент.
3. Пять лидирующих объектов (в обычной ситуации этих объектов может быть 100 или больше) отправляются на следующий этап – анализ и прогнозирование.
4. Прогноз составляется на основе предпочтений пользователя, информация о которых берется из базы данных о пользователе. При этом, скорее всего, один или несколько объектов будут исключены из списка вследствие прогнозируемо низкой оценки. На рис. 1.9 исключены объекты C и E.
5. В результате анализа и прогноза остаются наиболее значимые объекты, которые теперь сопровождаются прогнозируемыми оценками. Они переходят на следующий этап – выстраивание в порядке релевантности.
6. Релевантные объекты располагаются в соответствии с предпочтениями пользователя, контекстом и демографическими данными. Этот процесс может даже предполагать максимальное расширение разнообразия подборки.
7. Теперь объекты расставлены по релевантности. Объект F был исключен, поскольку анализ релевантности показал, что данный контент не релевантен интересам конечного пользователя.
8. Система выдает список.
9. Сервер выдает результат.

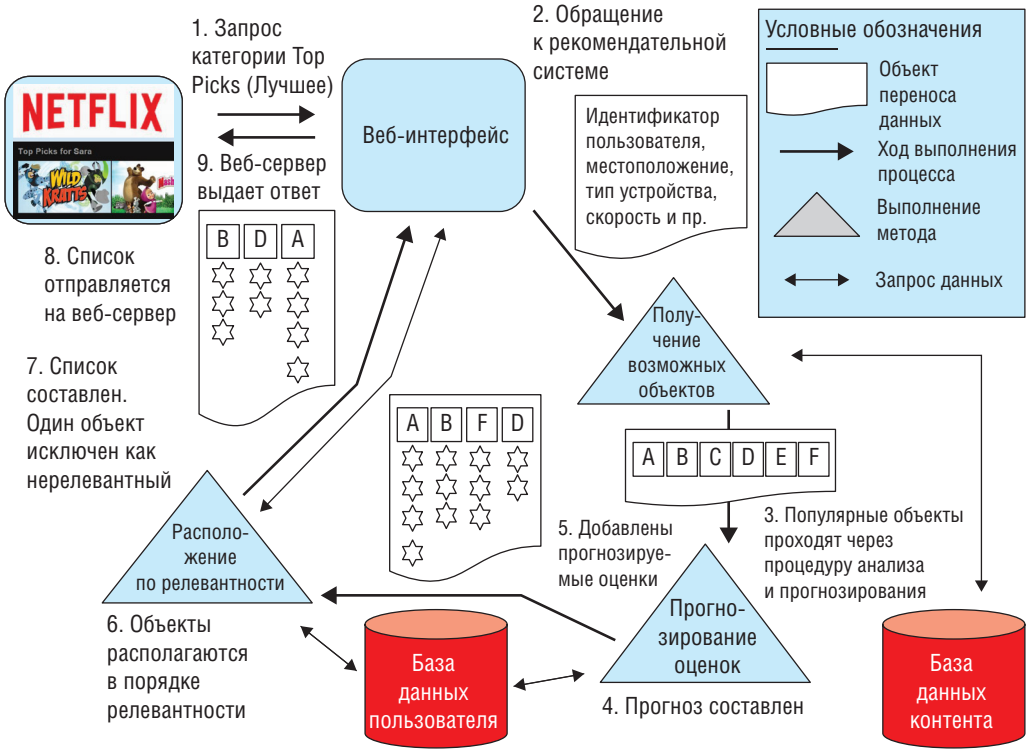


Рис. 1.9. Как может выглядеть процесс отбора контента в категорию Top Picks (Лучшее) на Netflix

Изучив рис. 1.9, легко понять, что при работе с рекомендательными системами необходимо учитывать множество аспектов. В этом описании отсутствуют этапы сбора данных и построения моделей. Большинство рекомендательных систем пытаются тем или иным образом применять данные, показанные на рис. 1.10.

Кроме того, рис. 1.9 иллюстрирует еще один факт, который необходимо учитывать: прогнозирование оценок/рейтингов – это всего лишь одна из множества задач рекомендательной системы. Другие факторы также могут оказывать существенное влияние на то, что именно система будет показывать пользователю. В данной книге очень много внимания уделяется прогнозированию оценок, и это важный аспект, даже если вам показалось, что я не придаю ему большого значения.

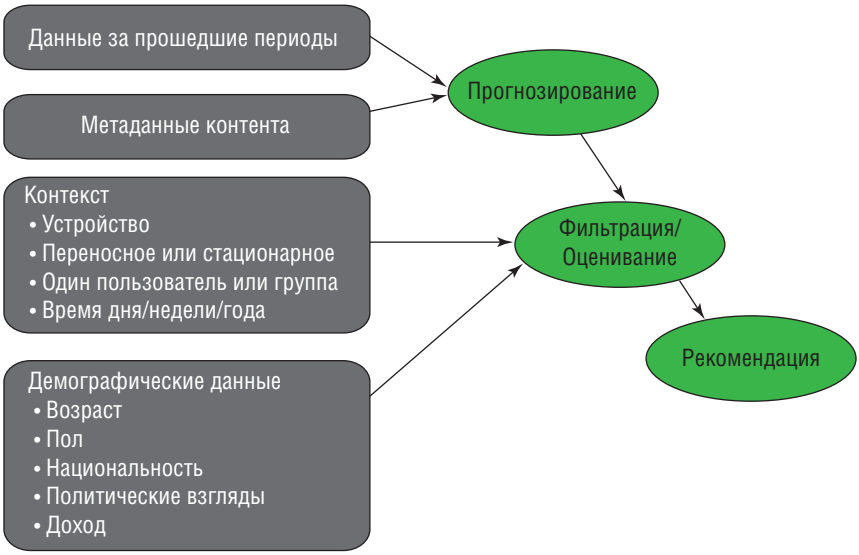


Рис. 1.10. Данные, на которые может опираться рекомендательная система

1.2. Таксономия рекомендательных систем

Прежде чем начать разработку рекомендательной системы, хорошо бы немного поразмыслить, какую именно рекомендательную систему вы хотите получить на выходе. Отличной отправной точкой будет изучение аналогичных систем. В этом разделе вы познакомитесь с основными понятиями, необходимыми для понимания рекомендательных систем.

В предыдущем разделе мы совершили обзорную экскурсию по сервису Netflix и получили общее представление о возможностях рекомендательной системы. В данном разделе мы поговорим об основных характеристиках и классификации рекомендательных систем. Эту информацию я изначально почерпнул из курса профессора Джозефа А. Констана (Joseph A. Konstan) и Майкла Д. Экстранда (Michael D. Ekstrand) под названием «Introduction to Recommender Systems» («Введение в рекомендательные системы»)[1], и она мне много раз пригождалась. *Таксономия* позволяет описать систему по следующим параметрам: специализация, задача, контекст, степень персонализации, чьи мнения, конфиденциальность и надежность, интерфейс и алгоритмы[2]. Давайте рассмотрим каждый из этих параметров.

---

[1] Подробнее об этом вы найдете по адресу **www.coursera.org/learn/recommender-systems-introduction/**.

[2] Этот вариант таксономии впервые был обозначен в книге Джона Риэля (John Riel) и Джозефа А. Констана (Joseph A. Konstan) «Word of Mouse: The Marketing Power of Collaborative Filtering» (Business Plus, 2002).

1.2.1. Специализация

Специализация – это тип рекомендуемого контента. В случае с Netflix специализация – это фильмы и телесериалы, но специализация может быть абсолютно любой: подборки контента в виде плейлистов, советы по применению цифровых обучающих курсов для достижения своих целей, работа, книги, машины, продукты, отдых, путешествия или даже знакомства.

Специализация играет важную роль, поскольку служит ориентиром при организации работы с рекомендациями. Специализация также важна, поскольку позволяет оценить, насколько плохими будут последствия ошибок. Если у вас музыкальная рекомендательная система, ничего страшного не произойдет, если вы порекомендуете неудачную музыку. Если вы рекомендуете опекунскую семью для ребенка в трудной жизненной ситуации, цена ошибки будет очень высока. Кроме того, от специализации зависит, можно ли рекомендовать одно и то же по нескольку раз.

1.2.2. Задача

Какова задача сайта Netflix – как с точки зрения конечного пользователя, так и с точки зрения провайдера? Конечным пользователям рекомендации Netflix нужны, чтобы найти подходящий контент, который будет интересно посмотреть в определенное время. Представьте себе, что весь контент предлагается без какой-либо фильтрации или упорядочивания. Как в таком случае можно было бы найти хоть что-нибудь в каталоге Netflix, если в нем более 10 000 наименований? А задача провайдера (в данном случае Netflix) – подтолкнуть пользователей месяц за месяцем платить за подписку, предлагая контент, который человек захочет посмотреть, избавляя пользователей от лишних телодвижений.

Для Netflix главным индикатором успешности системы является объем просмотренного контента. Когда мы оцениваем степень достижения цели, которая не является нашей непосредственной целью, мы говорим о вспомогательной цели. Ставя перед собой вспомогательную цель, необходимо соблюдать осторожность, поскольку все может закончиться тем, что вы начнете следить вовсе не за теми параметрами, которые вам были важны изначально. Если человек много времени проводит на Netflix, возможно, он просто растерян, поскольку ищет, ищет и все никак не может найти то, что ему нужно. А может, он нашел, что искал, но сайт «затормозил»[1].

Возможно также, что подспудно Netflix стремится сделать все так, чтобы за просмотренный вами контент сервису пришлось платить как можно меньше. Вероятно, Netflix меньше платит за первые сезоны сериала «Друзья», чем за более свежие серии. А еще лучше, если вы будете смотреть сериалы производства Netflix – в этом случае сервис не платит никаких отчислений вообще.

---

[1] Если вам интересно, что может пойти не так, когда вы ориентируетесь на вспомогательные цели, прочитайте книгу Кэти О'Нил (Cathy O'Neil) «Weapons of Math Destruction» (Broadway Books, 2016).

Задача также может заключаться в том, чтобы предоставить информацию, оказать помощь или просветить человека в каком-либо вопросе. Однако чаще всего задача – совершить как можно больше продаж.

На каких пользователей вы ориентированы в первую очередь: тех, которые обратятся к вам лишь однажды и будут ждать от вас хороших рекомендаций, или тех, которые зарегистрируются и будут заходить регулярно? Будет ли сайт загружать контент автоматически (как, например, радио на сервисе Spotify, где нон-стоп проигрывается музыка, выбор которой зависит от первой песни или исполнителя)?

1.2.3. Контекст

Контекст – это условия, в которых пользователь получает рекомендацию. В нашем примере это может быть устройство, с которого человек заходит на Netflix, или текущее местонахождение, время дня (или ночи), а также то, чем человек занят. Есть ли у пользователя время изучить предложенные рекомендации или ему нужно быстро принять решение? Контекст может также включать в себя погоду и даже настроение пользователя!

Возьмем, например, поиск кафе через Google Карты. Сидит ли человек в офисе за компьютером и ищет хорошую кофейню или стоит на улице, когда начинается дождь? В первом случае лучшей выдачей будет список хороших кофеен на достаточно обширной территории, во втором – рекомендация в идеале будет содержать только ближайшее место, где можно выпить кофе, пережидая дождь. Примером приложения, с помощью которого можно найти кафе, может быть Foursquare. О приложении Foursquare мы подробнее поговорим в главе 12.

1.2.4. Степень персонализации

Степень персонализации рекомендаций бывает самой разной, от применения наиболее обобщенных данных до изучения информации о конкретном пользователе. Эти уровни отображены на рис. 1.11.

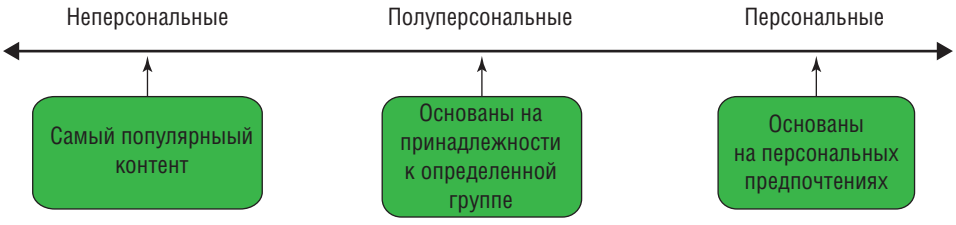


Рис. 1.11. Степени персонализации

Неперсональные

Список наиболее популярных объектов считается *неперсональной рекомендацией*: расчет здесь на то, что данному пользователю понравятся те же объекты, что и большинству других. К неперсональным рекомендациям также

относится выстраивание объектов в списке по дате их появления, например когда самыми первыми отображаются наиболее новые объекты. Любой человек, который взаимодействует с рекомендательной системой, видит те же рекомендации, что и остальные люди. Сюда же можно отнести спецпредложения в кафе, когда посетителям предлагаются алкогольные напитки в пятницу вечером, капучино по утрам и поздний завтрак утром в выходные.

Полуперсональные и частично персональные

Рекомендации следующего уровня предполагают деление пользователей по группам – это *полуперсональные* и *частично персональные рекомендации*. Пользователей можно разбить на группы по множеству разных признаков: по возрасту, по национальности или по характерному признаку – например, бизнесмены и студенты, водители автомобилей и велосипедисты.

В качестве примера: система, предназначенная для продажи билетов на мероприятия, рекомендует шоу-программы, основываясь на стране или городе пребывания пользователя. Другой пример: если пользователь слушает музыку на смартфоне, система может попытаться определить, перемещается это устройство или нет. Если да, то, возможно, человек занимается спортом или едет на машине или велосипеде. Если устройство неподвижно, пользователь, вероятно, сидит на диване дома и ему подойдет несколько иная музыка.

Подобная рекомендательная система не знает ничего лично о вас, для нее вы являетесь частью определенной группы или сегмента. Другие люди, входящие в эту группу, получат те же рекомендации, что и вы.

Персональные

Персональные рекомендации базируются на данных о конкретном пользователе, а именно на информации о том, каким образом пользователь взаимодействовал с системой ранее. Так формируются рекомендации специально для данного пользователя.

Большинство рекомендательных систем при составлении персональных рекомендаций также учитывают принадлежность пользователя к группе и популярность контента. Пример персональных рекомендаций можно увидеть на сайте Amazon, где в личном кабинете пользователя присутствует раздел **Recommended for You** (Рекомендуется вам). Главная страница сервиса Netflix – это ярчайший пример персональных рекомендаций.

Обычно сайты комбинируют различные типы рекомендаций. Лишь несколько сайтов, включая Netflix, предлагают только персональные рекомендации. На Amazon также мы увидим раздел **Most Sold Items** (Самые продаваемые товары), где отсутствует персонализация, а также раздел **Customers Who Bought This Also Bought This** (Вместе с этим покупают эти товары), т. е. *выборочные рекомендации*. Эти рекомендации предлагаются выборочно – например, тем людям, которые смотрят данный товар.

1.2.5. Чье мнение

Экспертные рекомендательные системы формируют блок рекомендаций на основе ассоциативных правил, составленных вручную специалистами. Эти системы рекомендуют хорошие вина, книги и подобные вещи и применяются в областях, где, как правило, необходимо быть экспертом, чтобы давать советы.

Однако дни экспертных систем практически на исходе, поэтому сейчас параметр *чьи мнения* утратил свою актуальность. Почти все сайты опираются на мнение большинства. Говорят, что у каждого правила есть исключения: несколько экспертных сайтов все же продолжают работать. Один из них – специализирующийся на вине сайт **www.vivino.com** (рис. 1.12), на котором представлены рекомендации сомелье. Vivino также применяет рекомендательную систему по винам. В 2017 году в приложение Vivino была добавлена рекомендательная система, которая помогает пользователю выбрать новое, соответствующее его вкусам вино, опираясь на предыдущие оценки этого пользователя[1].

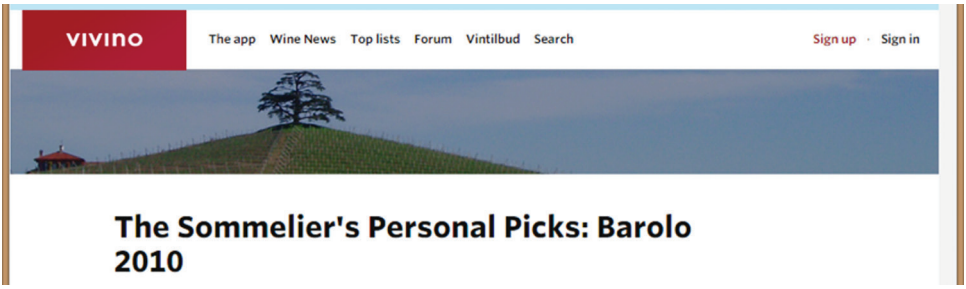


Рис. 1.12. Сайт **Vivino.com** предоставляет пользователям экспертные рекомендации по винам (сами рекомендации вырезаны для экономии пространства)

1.2.6. Конфиденциальность и надежность

Насколько хорошо система защищает данные о пользователе? Каким образом применяется полученная информация? Например, в Европе принято перечислять деньги на пенсионный счет, который находится в ведении банка. Часто эти банки предлагают различные накопительные пенсионные программы. Система, специализирующаяся на таких рекомендациях, должна очень строго следить за конфиденциальностью. Представьте себе, что вы заполняете заявление на участие в накопительной пенсионной программе, где указываете, что у вас проблемы со спиной, а через минуту после этого вам звонит мануальный терапевт с отличным предложением специально для решения вашей проблемы. Или еще хуже – вы покупаете специальную кровать для людей с больной спиной, а через час получаете электронное письмо, в котором говорится, что стоимость вашей медицинской страховки выросла.

[1] Подробнее читайте по ссылке **mng.bz/1jFR**.

Многие люди считают рекомендации своеобразной разновидностью манипуляций, поскольку с их помощью людям предлагают такие варианты товаров и услуг, на которые те с большей вероятностью согласятся, чем на предложения из случайной подборки. А большинство магазинов стремятся увеличить объем продаж. Тот факт, что благодаря рекомендациям магазины продают больше товаров и услуг, вызывает у людей ощущение, что ими манипулируют. Но если при этом просмотр фильма принесет удовольствие, а не навеет скуку, то я не возражаю. Манипуляция – это не столько сам факт того, что вам предлагают определенный объект, сколько *мотив*, для чего это *делается*. Если вам порекомендовали неподходящее и не самое оптимальное лекарство лишь на том основании, что его продавец предоставляет владельцу сайта лучшие условия сотрудничества, то это манипуляция, и это заслуживает порицания.

Когда рекомендательная система запущена и бизнес активно развивается, у многих может возникнуть соблазн продвинуть интересы поставщиков, быстрее сбыть залежалый товар или по каким-либо иным причинам подтолкнуть пользователей к покупке определенной торговой марки таблеток. Учтите: если пользователи почувствуют, что ими манипулируют, они перестанут доверять вашим рекомендациям и в итоге найдут то, что им нужно, где-нибудь в другом месте.

> *В тот момент, когда рекомендации обладают властью влиять на решение, они становятся мишенью для спамеров, мошенников и других лиц, желающих повлиять на наши решения из далеко не лучших побуждений.*
>
> Дэниел Тункеланг[1].

Надежность – это показатель того, насколько пользователь доверяет рекомендациям, в противовес тому, чтобы расценивать их как рекламу или попытки манипулирования. Говоря о Netflix, я рассказывал, что прогнозы могут отталкивать пользователей, если прогнозируемые оценки пользователя оказываются очень далекими от реальных. Все это – вопрос надежности. Если пользователь прислушивается к рекомендациям, система надежна, ей доверяют.

1.2.7. Интерфейс

Интерфейс рекомендательной системы отображает тип ввода и вывода данной системы. Давайте рассмотрим каждый из них.

Ввод

Пользователи сервиса Netflix с некоторых пор могут указывать, нравится им контент или не нравится, выставляя оценки и обозначая предпочитаемые жанры и темы. Эти данные могут служить в качестве ввода для рекомендательной системы.

---

[1] Чтобы больше узнать о том, какую роль играют доверие и вкус в рекомендациях, см. **www.linkedin.com/pulse/taste-trust-daniel-tunkelang**.

В примере с Netflix мы говорили о *явном вводе*, при котором вы, пользователь, вручную указываете информацию о том, что вам нравится. Другой вид ввода – *неявный*, когда система пытается установить ваши вкусы, опираясь на то, как вы с ней взаимодействуете. В главе 4 мы подробнее рассмотрим обратную связь.

Вывод

К разновидностям *вывода* относятся прогнозы, рекомендации или фильтрация. Например, Netflix выдает рекомендации разными путями. Сервис прогнозирует оценки, составляет персональный набор предложений и отображает популярный контент, как правило, в виде списка 10 лидирующих объектов (но даже этот список Netflix формирует персонально для данного пользователя).

Если рекомендации естественным образом вписаны в страницу, это *органичная подача*. Расположенные рядами категории контента в сервисе Netflix – это пример органичных рекомендаций: Netflix не указывает, что это рекомендации. Это обычная часть сайта.

На рис. 1.13 показаны неорганичные результаты. То, что мы видим на сайте Hot Network Questions, – это разновидность неперсонализированных рекомендаций, поскольку все приведенные утверждения воспринимаются пользователем так, будто они сделаны от имени сайта. Amazon предлагает неорганичные персонализированные рекомендации в разделе **Recommended for You** (Рекомендуется вам), а New York Times применяет неорганичные рекомендации, когда показывает наиболее часто отправляемые по электронной почте статьи.

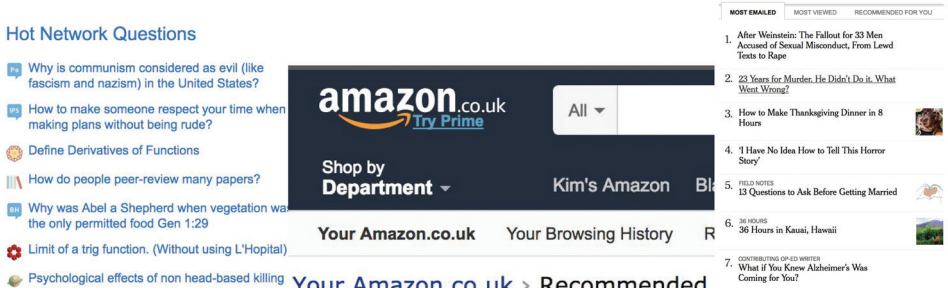


Рис. 1.13. Примеры неорганичных, неперсонализированных рекомендаций: раздел **Hot Network Questions** (Горячие вопросы в сети) с сайта Cross Validated, раздел **Most Emailed** (Наиболее часто пересылаемые) с сайта New York Times и персонализированный раздел **Recommended for You** (Рекомендуется вам) на сайте Amazon

Некоторые системы объясняют предоставленные рекомендации. Это системы, применяющие принцип «*белого ящика*». Те системы, которые свои

рекомендации не объясняют, построены на принципе «*черного ящика*». На рис. 1.14 показаны примеры каждого из типов. Это важное различие, которое необходимо учитывать при выборе алгоритма, поскольку не каждый алгоритм предусматривает возможность отследить процесс принятия решения до самых его истоков.

При выборе типа рекомендательной системы (построенной по принципу «белого ящика» или «черного ящика») нужно принимать во внимание особенности каждого из алгоритмов. Чем больше объяснений потребуется от системы, тем проще алгоритм. Часто решение можно проанализировать, как показано на рис. 1.15. Чем выше качество рекомендации, тем сложнее объяснение. Эта проблема известна как *компромисс между точностью модели и сложностью интерпретации модели*.

Как-то я работал над проектом, в котором огромное значение уделялось объясняемости и качеству. Чтобы справиться с задачей, пришлось надстроить еще один алгоритм поверх нашей рекомендательной системы. Это позволило нам получить рекомендации высокого качества, и при этом система была способна сопоставить использованные данные с результатом.

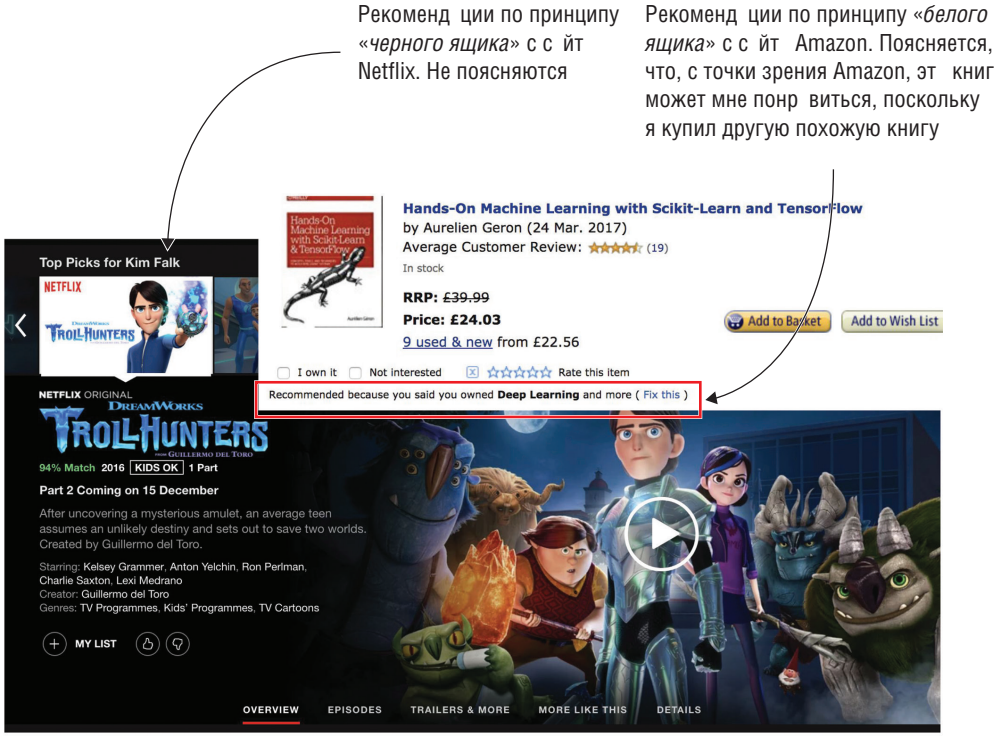


Рис. 1.14. Рекомендации по принципу «черного ящика» (с сайта Netflix) и «белого ящика» (с сайта Amazon)

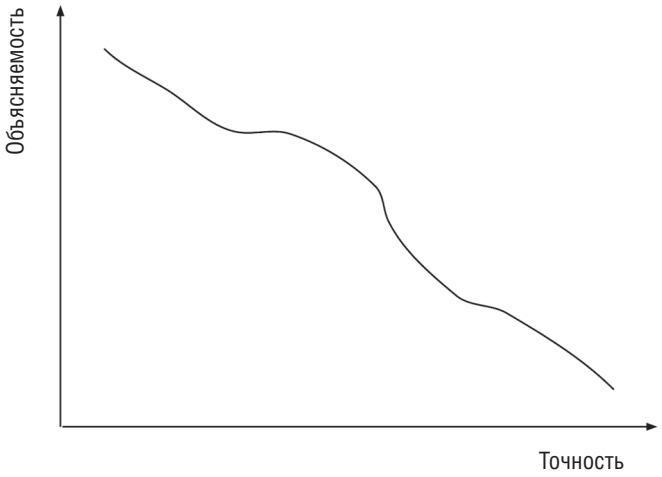


Рис. 1.15. Объясняемость и качество рекомендаций

В последнее время рекомендательные системы получили очень широкое распространение, поэтому в нашем распоряжении множество примеров. Чаще всего рекомендательные системы заточены под фильмы, музыку, книги, новости, исследовательские статьи и вообще под большинство товаров. Но рекомендательные системы применяются также и во многих других сферах, включая финансовые услуги, страхование жизни, цифровые данные, поиск работы, т. е., по сути, везде, где требуется сделать выбор. В этой книге в качестве примеров приводятся в основном сайты, но с другими платформами тоже вполне можно работать.

1.2.8. Алгоритмы

В этой книге приводится несколько алгоритмов. Алгоритмы можно разделить на две группы по типу данных, на основе которых они формируют рекомендации. Алгоритмы, которые опираются на данные о предыдущих сеансах работы пользователя с системой, называются *совместной фильтрацией*. Алгоритмы, которые при составлении рекомендаций анализируют метаданные контента и пользовательские профили, называются *контентной фильтрацией*. Сочетание этих двух типов образует гибридные рекомендательные системы.

Совместная фильтрация

На рис. 1.16 представлен один из способов реализации совместной фильтрации. Внешний контур – это весь каталог. Контур посередине – это группа людей, которые приобрели/воспользовались одними и теми же объектами. Рекомендательная система рекомендует объекты из малого овала (на переднем плане) исходя из того, что если пользователям нравится то же самое, что и текущему пользователю, то текущему пользователю понравятся и осталь-

ные объекты, заинтересовавшие данную группу. Группа определяется путем поиска соответствий между тем, что понравилось отдельным пользователям, и тем, что понравилось текущему пользователю. В этом случае в рекомендации попадет тот пласт контента, который упускает текущий пользователь (та часть средней окружности, которая не попадает в овал, включающий заинтересовавшие текущего пользователя объекты).

Существует множество способов формирования рекомендаций на основе совместной фильтрации. Простой способ описан в главе 8, а способ посложнее – в главе 11, где мы поговорим об алгоритмах факторизации матриц.

Рис. 1.16. Схема совместной фильтрации

Контентная фильтрация

Контентная фильтрация опирается на метаданные объектов из вашего каталога. Например, Netflix пользуется описаниями фильмов. В зависимости от алгоритма система может формировать рекомендации путем подбора объектов, аналогичных тем, которые понравились пользователю ранее, путем сопоставления объектов с данными из пользовательского профиля, либо, если пользователь не задействован, путем поиска схожих объектов из всего контента. При наличии пользовательского профиля система анализирует каждый профиль, в котором указаны категории контента. Если бы сервис Netflix применял контекстную фильтрацию, он мог бы составлять пользовательские профили с разбивкой по жанрам – например, триллеры, комедии, драмы и новинки – и указывать рейтинг каждого из жанров. В этом варианте фильм попадает в рекомендации, только если его рейтинг соответствует рейтингу, указанному пользователем.

Приведем пример. Пользователю Томасу понравились фильмы «Стражи галактики», «Интерстеллар» и сериал «Игра престолов». Каждому он дал оценку по пятибалльной шкале. В табл. 1.2 показан один из способов интерпретации этих оценок.

Таблица 1.2. Система оценок на примере двух фильмов и телесериала

Фильмы и сериалы	Научная фантастика	Приключения
«Интерсталлар»	3	3
«Игра престолов»	1	5
«Стражи галактики»	5	4

С опорой на эту информацию составляется профиль Томаса, в котором научной фантастике дана оценка 3, а приключениям 4. Для подбора и рекомендации других фильмов просматривается каталог и выделяются те фильмы, которые соответствуют профилю Томаса.

Гибридная рекомендательная система

И у совместной, и у контентной фильтрации есть сильные и слабые стороны. Для нормального функционирования совместной фильтрации от пользователей должна стабильно поступать обратная связь, а контентная фильтрация подразумевает наличие подробного описания у каждого объекта. Часто рекомендации формируются на основе результатов работы этих двух алгоритмов и анализа данных другого типа, например удаленность от какой-либо локации или время суток.

1.3. Машинное обучение и Netflix Prize

Рекомендательная система предназначена, для того чтобы предугадывать, какой контент нужен пользователю в данный момент. Предугадать это можно множеством различных способов. Создание рекомендательной системы превратилось в междисциплинарное мероприятие, в котором огромную роль играют различные информационные технологии, включая машинное обучение, интеллектуальный анализ данных, поиск информации и даже человеко-компьютерное взаимодействие. Машинное обучение и интеллектуальный анализ данных позволяют вычислительной машине строить прогнозы на основе изучения примеров того, что прогнозируется. Следовательно, эти же возможности прогнозирования могут участвовать в формировании рекомендаций.

Многие рекомендательные системы построены вокруг алгоритмов машинного обучения, направленных на прогнозирование оценки, которую пользователь поставит объекту, или на то, чтобы составить наиболее подходящий для данного пользователя порядок расположения объектов. Одна из причин активного развития направления машинного обучения заключается в том, что с его помощью специалисты пытаются решить проблему рекомендательных систем. Они стремятся получить действующий алгоритм, с помощью которого компьютер мог бы угадывать наши тайные желания еще до того, как мы сами поняли, чего хотим.

Многие утверждают, что апогеем интереса к внедрению технологий машинного обучения в работу рекомендательных систем стало знаменитое соревнование Netflix Prize. Соревнование проводил сервис Netflix, пообещав-

ший выплатить 1 млн долларов каждому, кто сумеет разработать алгоритм, который улучшит рекомендации сервиса на 10 %. Соревнование началось в 2006 году, а победитель появился только через три года. В итоге выиграл гибридный алгоритм. Помните, мы говорили, что гибридный алгоритм запускает несколько алгоритмов, а затем объединяет полученные результаты и выдает общий итог? О гибридных алгоритмах мы поговорим в главе 11.

Netflix так и не применил ставший победителем алгоритм. Вероятно, причина в том, что он был настолько сложным, что неоправданно сильно возрастала нагрузка на систему. Поэтому, к сожалению, Netflix ничем не поможет в нашем рассказе об этом аспекте рекомендательных систем. Вместо этого я разработал небольшой демосайт под названием MovieGEEKs, на примере которого буду объяснять описываемые в данной книге вещи. Над сайтом необходимо как следует потрудиться, чтобы привести его в рабочее состояние. Его главная задача – это демонстрация основных принципов работы рекомендательных систем.

1.4. Интернет-сайт MovieGEEKs

Эта книга рассказывает о том, как построить рабочую рекомендательную систему. Она вооружит вас необходимыми для этого средствами независимо от того, на какой платформе будет работать ваша рекомендательная система. Но для того чтобы придумать какую-нибудь интересную рекомендательную систему, необходимо собрать данные и понять, как все устроено, а для этого недостаточно просто посмотреть на цифры.

В центре внимания данной книги прежде всего сайты, однако это не означает, что все написанное здесь неприменимо к любому другому типу системы. Это краткое вступление подводит нас к тому фреймворку, от которого мы будем плясать.

Интернет-сайт MovieGEEKs (**mng.bz/04k5**) с помощью фреймворка Django. Я советую вам скачать MovieGEEKs и обращаться к нему по ходу чтения книги, поскольку так вам проще будет понять, о чем идет речь. Тот факт, что при создании сайта применялся фреймворк Django, не имеет большого значения. Приводя какие-либо примеры, я буду подсказывать вам, на что надо обратить внимание.

> **Интернет-сайт и фреймворк Django**
>
> Если словосочетание фреймворк Django вам ни о чем не говорит, почитайте документацию по Django на странице **www.djangoproject.com/start/overview/**.

Зайти нужно только на один этот интернет-сайт. Он содержит в себе весь функционал, описанный в данной книге. Он отражает именно тот вымышленный сценарий, которого мы будем придерживаться.

Представьте себе, что у вас есть клиент, который хочет продавать DVD-диски через интернет. Мне сразу приходит в голову старый магазинчик с прокатом DVD в английском городке Бате, владелец которого хочет попробовать продавать фильмы через интернет. К сожалению, этого магазина больше нет (рис. 1.17).

Магазин ни с какой точки зрения нельзя было назвать электронным. Учет дисков велся с помощью бумажных карточек, и – хоть вам, возможно, это покажется невероятным – система работала! В реальной жизни вряд ли владелец когда-либо смог перевести свой бизнес в интернет-среду, но одной из уникальных особенностей этого места было то, что посетители всегда получали там отличные рекомендации. Владелец составлял ежемесячные обзоры – рекомендации на основе экспертного мнения, – и люди, которые там работали, всегда знали о фильмах все.

Мне хочется верить, что рекомендательная система – это попытка предложить индивидуальный подход людям в интернете. Ниже в краткой форме изложены пожелания нашего выдуманного владельца.

Рис. 1.17. Витрина вымышленного магазина On the Video Front

1.4.1. Оформление и характеристики

Для начала необходимо обозначить несколько базовых идей оформления. На главной странице сайта должно присутствовать следующее:

- область с обложками фильмов;
- обзор каждого фильма, для прочтения которого не требуется переходить на следующую страницу;
- рекомендации, как можно более персонализированные;
- меню со списком жанров.

Для каждого фильма нужна собственная страница, на которой должны быть:

- киноафиша;
- описание;
- рейтинг.

Для каждой категории должна быть выделена собственная страница, отображающая:

- ту же структуру, что и главная страница;
- рекомендации по данной категории.

1.4.2. Архитектура

Для разработки данного сайта будет применяться язык программирования Python и фреймворк Django. Django позволяет разбить проект на несколько отдельных приложений. На рис. 1.18 показана высокоуровневая архитектура и дан пример того, какие приложения участвуют в создании сайта.

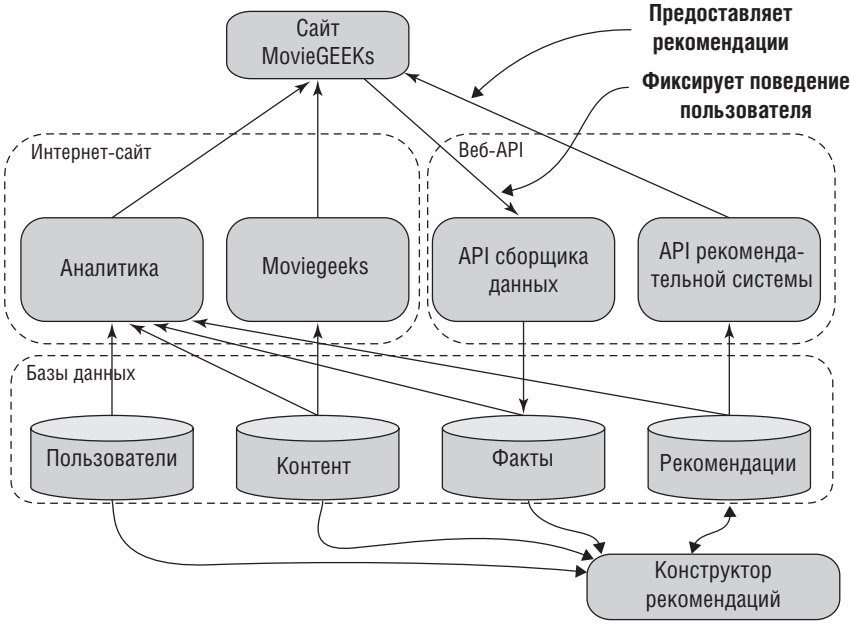


Рис. 1.18. Архитектура сайта MovieGEEKs

Давайте по-быстрому пробежимся по основным моментам:

- *MovieGEEKs* – это основная часть сайта. Здесь клиентская логика (HTML, CSS, JavaScript), наряду с программным кодом на языке Python, ориентирована на получение данных о фильмах.
- *Аналитика* – это капитанский мостик, откуда можно контролировать все процессы. Эта часть опирается на данные из всех баз. Об аналитике рассказывается в главе 4.
- *Сборщик данных* – отвечает за отслеживание шаблонов поведения пользователя и сохраняет их в базу фактов. Журнал фактов описан в главе 2.
- *Рекомендации* – сердце всей этой системы, без которого существование сайта было бы бессмысленно. Отсюда рекомендации поступают на сайт MovieGEEKs. Об этой части рассказывается в главе 5 и далее в книге.
- *Конструктор рекомендаций* – занимается предварительным составлением рекомендаций, на базе которых формируются тщательно проработанные рекомендации для пользователя. Разбираться с конструктором рекомендаций мы начнем в главе 7.

Каждый из этих компонентов или приложений содержит интересные модели данных и функции. Это станет наживкой для будущих посетителей.

MovieGEEKs – это сайт с фильмами, в основном потому что в нашем распоряжении имеется большой набор данных и контента, касающихся фильмов, пользователей и рейтингов. Более того, этот контент включает в себя URL-адреса, содержащие киноафиши, а с таким контентом работать очень приятно.

На рис. 1.19 показана главная страница сайта MovieGEEKs, т. е. целевая страница. Когда пользователь щелкает по значку фильма, появляется всплывающее окно с дополнительной информацией и ссылкой на более подробное описание.

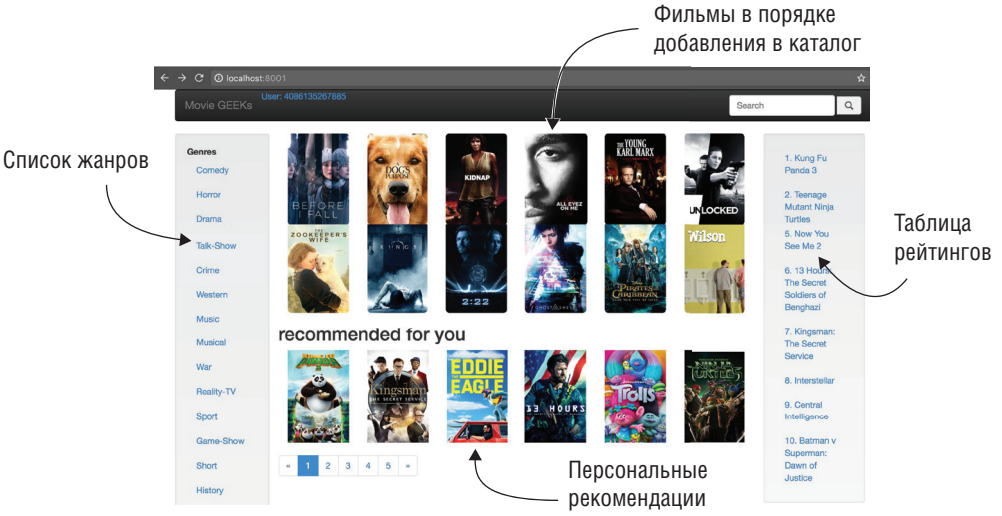


Рис. 1.19. Главная страница сайта MovieGEEKs

Вот и все! Просто, но эффективно. Скачайте материалы прямо сейчас. Инструкции по установке вы найдете в readme-файле на сервисе GitHub по ссылке **mng.bz/04k5**. Сайт MovieGEEKs опирается на набор данных под названием MovieTweetings. В этот набор данных входят оценки и рейтинги фильмов, полученные из качественных записей в Twitter[1].

1.5. Создание рекомендательной системы

Прежде чем двигаться дальше, давайте посмотрим, как создается рекомендательная система. Предположим, что у вас уже есть готовая платформа – например, сайт или приложение – и вам нужно просто добавить туда рекомендательную систему. В этом случае весь процесс будет выглядеть примерно так, как показано на рис. 1.20.

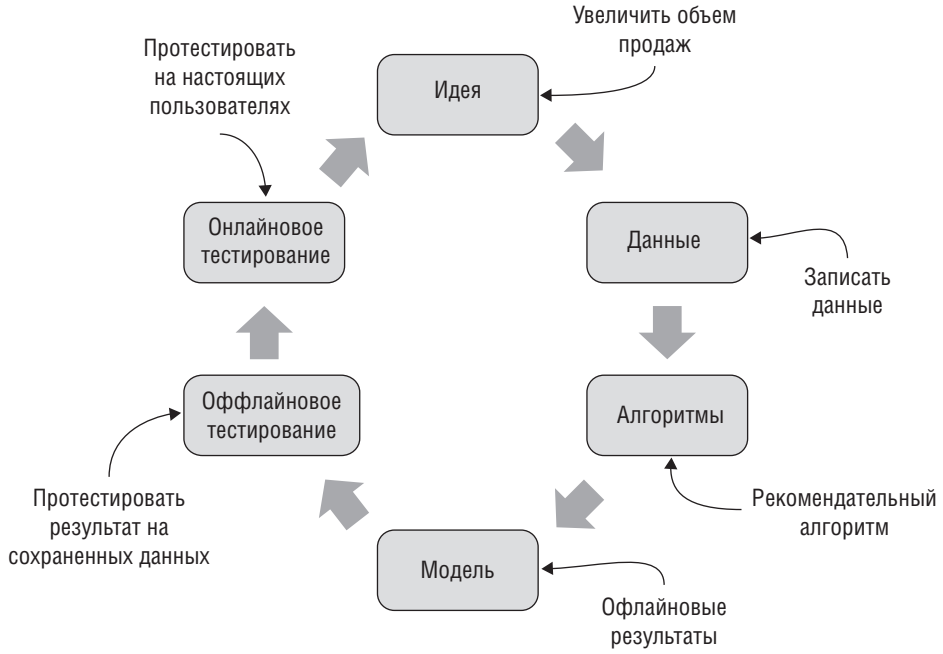


Рис. 1.20. Ориентированный на данные подход к созданию рекомендательной системы

Сначала у вас появляется желание увеличить объем продаж и возникает идея, что достичь этого можно с помощью рекомендательной системы. Вы начинаете собирать данные о поведении пользователей и на основе этих данных создаете алгоритм, при выполнении которого запускается модель. Эту модель можно также считать *функцией*, которая, получив идентификационные данные пользователя, просчитает рекомендации.

Вы опробуете эту модель на ретроспективных данных, чтобы проверить, можно ли с их помощью спрогнозировать поведение пользователя в буду-

[1] Подробнее читайте по ссылке **github.com/sidooms/MovieTweetings**.

щем. Например, если у вас есть данные о покупках пользователей в прошлом месяце, вы можете создать модель на основе данных за первые три недели и посмотреть, насколько хорошо она спрогнозирует поведение пользователя за последующие три недели того месяца, за который у вас собраны данные. Возможно, такая модель предугадает, какие товары купили пользователи, даже точнее, чем это сделала бы базовая рекомендательная система, которая может просто выполнять метод, выдающий наиболее популярные товары. В случае успеха можно испытать этот метод на группе пользователей и проверить, есть ли изменения к лучшему. Если есть, тогда метод работает и может применяться, в ином случае необходимо вернуться назад и доработать его.

Теперь вы уже должны понимать, что такое рекомендательная система, в каких данных она нуждается и что она может дать на выходе. Зная основные принципы работы рекомендательных систем, можно переходить к главе 2, в которой говорится о том, как собирать данные о пользователях.

Резюме

- Сервис Netflix с помощью рекомендаций обеспечивает персональный подход к каждому пользователю и помогает пользователям находить контент, который им понравится.
- В широкое понятие «рекомендательная система» входит множество различных компонентов и методов.
- Прогноз и рекомендация – это не одно и то же. Прогнозирование – это попытка угадать, какую оценку пользователь даст тому или иному контенту, а рекомендация – это список объектов, соответствующих вкусам пользователя.
- Контекст рекомендации – это все, что происходит вокруг пользователя (окружающая пользователя обстановка) в момент формирования рекомендации. Объекты, которые, возможно, по прогнозу, получат не самые высокие оценки, могут попасть в рекомендации, если вписываются в контекст.
- Описанная в данной главе таксономия может пригодиться, когда вы изучаете другие рекомендательные системы или пытаетесь разработать собственную. К данной таксономии не мешает обратиться, перед тем как начинать работу над своей рекомендательной системой.

Глава 2

Поведение пользователя, и как собирать о нем данные

В этой главе вы познакомитесь с интересной темой – сбором данных:

- сначала вы вернетесь на сайт Netflix и изучите события, которые могут считаться опорными фактами для построения выводов о вкусах и интересах пользователя;
- вы узнаете, как создать сборщик данных для объединения информации обо всех этих событиях;
- вы узнаете, как добавить подобный сборщик данных на сайт, например MovieGEEKs, чтобы он обрабатывал события вроде тех, которые мы наблюдали на сайте Netflix;
- получив общую информацию и выполнив ряд необходимых действий, вы остановитесь и проанализируете поведение пользователей в целом.

Факты – это данные, отражающие вкусы пользователя. Когда мы говорим о сборе фактов, имеется в виду сбор информации о событиях и действиях, которые служат ключами к пониманию вкусов пользователя.

В большинстве книг о рекомендательных системах описываются алгоритмы и способы оптимизации этих алгоритмов. На старте у вас уже должен быть большой объем данных, необходимых для наполнения алгоритма. Подобный набор данных у вас будет и для сайта MovieGEEKs. В эти данные входит каталог фильмов и оценок реальных пользователей. Но эти данные не возникают сами собой, как по волшебству. Для того чтобы собрать необходимый набор фактов, требуется немало времени и усилий. От этого зависит успех или провал вашей системы. Известная поговорка программистов «Garbage in, garbage out» (Мусор на входе – мусор на выходе) актуальна и для рекомендательных систем.

К сожалению, данные, которые подходят для одной системы, *могут* не подойти для другой. По этой причине мы серьезно отнесемся к разговору о том, какие данные вам могут пригодиться, но я не гарантирую, что в вашей среде все будет работать именно так, как здесь описано.

В этой главе и в книге в целом мы рассмотрим множество вариантов сбора данных для собственного сайта. Как правило, от пользователей системы получают обратную связь двух типов: *явную* (оценки или лайки) и *неявную* (активно фиксируется в процессе наблюдения за пользователем). Пользователь может дать явную обратную связь в виде выставления определенного количества звезд, шляп, смайликов или любых других значков, указывающих, в какой степени ему понравился объект. Обычно оценки выставляются по пятибалльной шкале (или десятибалльной). Оценки пользователя – это, как правило, первое, что приходит в голову, когда речь идет о фактах. Мы будем обсуждать оценки далее в этой главе, но это не единственный ключ к предпочтениям пользователя.

Книга Рона Закарски (Ron Zacharski) «A Programmers Guide to Data Mining» (Руководство по поиску и анализу данных для программистов) – это отличное пособие, в котором объясняется различие между явными и неявными фактами[1]. Он приводит пример явной обратной связи со стороны человека по имени Джим. Джим указал, что является вегетарианцем и любит французские фильмы, но в его кармане лежит чек за прокат фильма «Мстители» от Marvel и чек за покупку упаковки из 12 банок пива Pabst Blue Ribbon. На основе какой информации вы будете формировать рекомендации? С моей точки зрения, выбор очевиден. Как по-вашему, что Джим ожидает увидеть в рекомендациях, открывая сайт со списком местных ресторанов, предлагающих еду на вынос: вегетарианскую еду или фаст-фуд? Собирая данные о поведении пользователей, можно понять, что нужно пользователям вроде Джима.

Вы поймете, что надежные факты ничем заменить нельзя. Давайте посмотрим, какие данные мог бы извлечь Netflix, и как он мог их истолковать.

2.1. Как (по моему мнению) Netflix собирает факты, пока вы пользуетесь сервисом

Говоря о фактах, снова вернемся к сервису Netflix. Все, что находится на главной странице Netflix, персонализировано (заголовки категорий и их содержимое), за исключением верхней категории, где размещается реклама, которую, вероятно, видят все пользователи или – также возможно – пользователи, похожие на меня. На рис. 2.1 показана моя персонализированная страница Netflix.

Каждый пользователь видит свои категории, заголовки варьируются от стандартных описаний жанров, например **Комедии** и **Драмы**, до максимально подогнанных под интересы человека названий – например, Фантастические фильмы про путешествия во времени 1980-х годов[2]. Первые несколько категорий на моей странице содержат новый контент, предложения и популярный контент. На тот момент, когда был сделан этот снимок экрана, пер-

[1] Книгу можно бесплатно скачать со страницы **guidetodatamining.com**.

[2] Этот пример приводит сам Netflix. Мне было бы интересно узнать, что входит в эту категорию, поскольку я уверен, что захотел бы посмотреть все эти фильмы, но кроме трилогии «Назад в будущее» мне ничего в голову не приходит. Подробнее о любопытных категориях Netflix читайте в статье по ссылке **mng.bz/xCvj**.

вая персонализированная категория – это категория под названием **Dramas** (Драмы), из чего можно сделать вывод, что, по мнению Netflix, из всех жанров (категорий) именно драмы заинтересуют меня в первую очередь. В категории **Dramas** отображается тот материал, который Netflix считает наиболее интересным для меня в этой категории. На рис. 2.2 показано, какой контент Netflix включил в список подобранных для меня драм.

Первая строка с персонализированным контентом

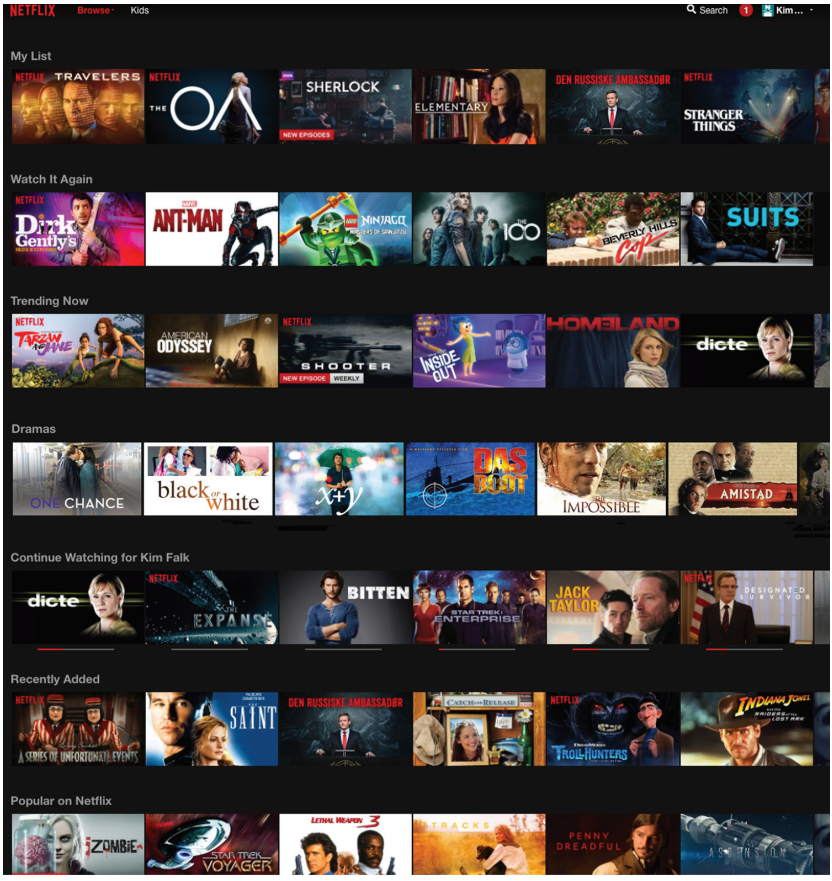


Рис. 2.1. Моя персонализированная главная страница в сервисе Netflix. В данный момент на первом месте стоит категория **Dramas** (Драмы)

Если навести курсор на категорию, можно прокрутить ее вправо или влево и посмотреть другие предложенные фильмы в обозначенном жанре. Что можно понять обо мне, когда я прокручиваю контент в категории **Dramas**? Это может обозначать, что я пишу книгу и пытаюсь сделать снимки экрана, но с большей степенью вероятности из этого следует, что я люблю драмы и хочу посмотреть, что еще есть в этой категории.

Если я вижу фильм, который кажется мне интересным, я могу навести на него курсор и прочитать описание. Более того, если это описание меня заинтриговало, я могу щелкнуть по нему и перейти на страницу фильма. Если мне

по-прежнему все нравится, я либо добавляю его в свой список, чтобы посмотреть позже, либо начинаю смотреть сразу.

Можно возразить, что на любом коммерческом сайте подробное изучении информации обозначает, что человек заинтересовался, сам факт просмотра фильма (до конца) – это эквивалент покупки товара. Но когда речь идет о стриминговом сервисе, на этом этапе рано ставить точку. Если посетитель начинает смотреть фильм, это хорошо, но, если он останавливает просмотр на четвертой минуте и больше не возвращается к фильму, это говорит о том, что ему не понравилось увиденное. Если пользователь пересматривает фильм позже или через какое-то время, значит, ему понравился этот фильм – возможно, даже больше, чем при первом просмотре.

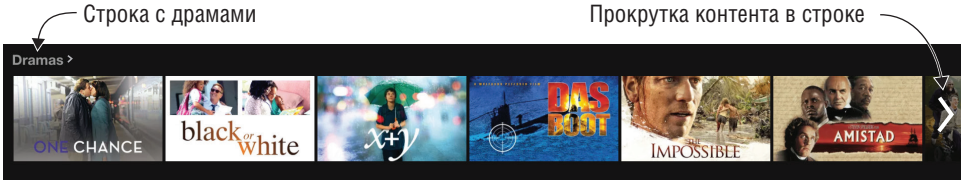


Рис. 2.2. Категория **Dramas** (Драмы) сервиса Netflix, в которой можно прокручивать контент с помощью стрелок или щелкнуть по названию категории и увидеть весь список драматических фильмов

Часто оказывается, что задача совсем в другом

Часто перед коммерческими сайтами стоит задача заставить людей купить товары, даже если это не совсем те товары, которые им нужны. Тут все зависит от того, насколько тесно сайт связан с товаром. Если на сайте Amazon вы купили футболку низкого качества, вы решите, что виноват производитель футболки. Но чтобы купить другую футболку, вы вполне можете вернуться на сайт Amazon. Если вы купили футболку, скажем, на официальном сайте Gap, и ее качество вам не понравилось, вы, вероятно, в следующий раз будете искать футболку на другом сайте.

Сайты, которые зарабатывают на подписке, работают иначе. Mofibo (**mofibo.com**), книжный стриминговый сервис на датском, шведском и голландском языках, предлагает рекомендации как источник вдохновения и новых открытий, но с описанием – для Mofibo важно, чтобы читатель знал, что это за книга, прежде чем начнет ее читать. Дело в том, что Mofibo выплачивает отчисления каждый раз, когда читатель открывает новую книгу (не за страницу, а за всю книгу в целом), и, хотя сервис Mofibo заинтересован в том, чтобы вы читали как можно больше, он также стремится сократить количество книг, за которые ему придется платить.

2.1.1. Какие факты собирает Netflix

Давайте попробуем представить, что происходит в кулуарах Netflix и какие данные собирает этот сервис. Допустим, сейчас вечер субботы и Джимми сидит дома. Пользовательский идентификатор Джимми в сервисе – 1234. Он приго-

товил себе попкорн в микроволновке, загрузил Netflix и сделал следующее:

- прокрутил контент в категории с драмами – **Dramas** (идентификатор 2);
- навел курсор на один из фильмов (идентификатор 41335), чтобы прочитать описание;
- щелкнул курсором, чтобы получить более подробное описание фильма (идентификатор 41335);
- начал смотреть этот фильм (идентификатор 41335).

Пока он смотрит фильм, попробуем представить, что происходит на сервере Netflix. В табл. 2.1 указано несколько событий, которые могли заинтересовать систему при анализе поведения данного пользователя, а также вероятные трактовки этих событий. Кроме того, я добавил столбец с названиями событий, чтобы было видно, как данные в табл. 2.1 связаны с журналом, о котором говорится позже.

Таблица 2.1. Примеры событий на сервисе Netflix

Событие	Значение	Название события
Прокручивание тематической категории	Пользователю интересна эта тема, в данном случае – драмы	`genreView`
Наведение курсора на фильм для просмотра краткого описания	Пользователя заинтересовал фильм (драма), что подтверждает его интерес к категории в целом	`details`
Щелчок по фильму для просмотра подробного описания	Пользователя еще сильнее заинтересовал фильм	`moreDetails`
Добавление фильма в список **My List** (Мой список)	Пользователь хочет посмотреть фильм позже	`addToList`
Начало просмотра фильма	Пользователь «покупает» фильм	`playStart`

Как видно из табл. 2.1, все эти события воспринимаются системой как опорные факты, поскольку они помогают выявить интересы пользователя. Таблица 2.2 показывает, в каком виде эти факты мог зафиксировать сервис Netflix.

Таблица 2.2. Как Netflix записывает факты (вероятный вариант)

Идентификатор пользователя	Идентификатор контента	Событие	Дата
1234	2	`genreView`	2017-06-07 20:01:00
1234	1234 41335	`details`	2017-06-07 20:02:21
1234	1234 41335	`moreDetails`	2017-06-07 20:02:30
1234	1234 41335	`addToList`	2017-06-07 20:02:55
1234	1234 41335	`playStart`	2017-06-07 20:03:01

Вероятно, в реальной жизни здесь был бы длинный список других столб-

цов, таких как тип устройства, место, время и погода, – вся эта информация может стать ключом к пониманию того, какой контекст окружает пользователя. Я даже рискну предположить, что количество журнальных записей в этом случае было бы гораздо выше, но давайте не будем излишне нагружать наш пример деталями. Список типов событий в реальности, вероятно, также был бы гораздо длиннее.

Теперь, когда вы получили общее представление о том, что такое факты в понимании рекомендательной системы, можно начинать разбираться с тем, как наладить работу сборщика этих фактов. *Сборщик фактов* применяется, для того чтобы собирать данные, указанные в табл. 2.2 и подобные. Чтобы вы не решили, что все это относится только к стриминговым сайтам, рассмотрим другой возможный сценарий.

Пример с сайтом по продаже садового инвентаря

Когда-то у меня был коллега, который все свое свободное время тратил на то, чтобы найти в интернете какой-нибудь садовый трактор или еще что-нибудь, работающее на бензине и пригодное для работы в саду. Представим себе, что у этого моего бывшего коллеги выдалась свободная минутка и он открывает свой любимый (вымышленный) сайт под названием Super Power Garden Tools (Сверхмощный садовый инвентарь). Он сделал следующее:

- выбрал категорию **Садовый трактор;**
- щелкнул по зеленому монстру, который способен перетаскивать деревья;
- щелкнул по перечню характеристик, чтобы посмотреть, деревья какого размера эта штука может перетаскивать;
- купил этого зеленого монстра.

Эти события аналогичны процессу выбору фильма. Пусть даже покупка дорогого товара – это более значительное действие, чем просмотр фильма, но я надеюсь, вы уловили суть.

2.2. Поиск полезных данных о поведении пользователя

Сайты, на которых пользователи проявляют высокую активность, позволяют своим владельцам получать большие объемы релевантных данных. Что касается сайтов, которые большинство пользователей посещает один раз, то тут необходимо в первую очередь обратить внимание на то, с каким контентом пользователь взаимодействовал и как весь этот контент связан между собой. Не отчаивайтесь, если вы не владелец стримингового сервиса, где пользователи очень активны и оставляют много данных о себе. Высока вероятность, что и на вашем сайте есть где разгуляться (в плане сбора данных).

В идеале рекомендательная система должна собирать все данные о пользователе в момент, когда тот взаимодействует с контентом – вплоть до оценки мыслительной активности, уровня адреналина в крови при изучении того или иного объекта, а также того, насколько сильно потеют ладони человека.

Чем больше растет наше присутствие в интернете, тем более реалистичным становится этот сценарий.

В мультфильме «ВАЛЛ-И» люди деградировали до состояния бесформенных существ, проводящих всю жизнь в кресле напротив монитора, а вся информация о них поступает в компьютер. (Задумайтесь об этом на минутку. Я провожу большую часть времени, глядя в монитор. Но я хотя бы перемещаюсь от одного монитора к другому.) Поскольку у большинства людей есть и другие дела, помимо того чтобы быть объектом изучения со стороны рекомендательной системы, нам нужно немного снизить свои ожидания. Но в сети мы стали ближе к пользователям, чем когда-либо прежде в обычных магазинах, поэтому у нас появилась возможность узнать много нового.

Влияние контента на отношение к системе в целом

В главе 1 мы говорили, что один из таксономических параметров – это задача. Задача важна, поскольку она определяет возможную стратегию формирования рекомендаций, а также список объектов, которые вы будете рекомендовать.

Возьмем, к примеру, фильм: если вы посмотрели плохой фильм на Netflix, вы делаете определенный вывод о качестве контента на Netflix и, соответственно, недовольны самой площадкой Netflix. Если на Amazon продается Blu-ray-диск с плохим фильмом, вряд ли ваше отношение к самому магазину Amazon изменится к худшему. А вот если вы ищете там этот диск с фильмом и не находите, тогда, вероятно, ваше мнение об Amazon станет хуже. Я рассказывал об этом в разделе 2.1, когда приводил пример с футболками, но, как говорится, повторенье – мать ученья.

Задача Netflix заключается в том, чтобы показывать хорошие фильмы, которые вам понравятся. Amazon показывает товары, которые можно купить, а понравились они вам или нет – это не так важно. Amazon тратит много ресурсов на то, чтобы побудить покупателей написать отзыв и оценить товар, поэтому не совсем честно будет сказать, что ему все равно. Но для большей наглядности я скажу именно так.

2.2.1. Как узнать мнение посетителя

Чтобы лучше описать события, которые возникают при взаимодействии покупателя с товаром, я условно разделил это взаимодействие на несколько этапов. Эти этапы показаны на рис. 2.3:

1. Покупатель смотрит. Как и в обычном магазине, покупатель осматривается, чтобы понять, что тут есть, без какой-либо конкретной цели. На этом этапе нужно обращать внимание на то, где останавливается покупатель и к чему проявляет интерес.
2. Покупатель заинтересовался одним или несколькими товарами. Может быть, он знал с самого начала, что ему нужно, и искал именно это, а может быть, товар ему понравился случайно.
3. Покупатель добавляет товар в корзину или список, намереваясь ку-

пить его.
4. Покупатель приобретает товар.
5. Покупатель пользуется товаром. Например, он смотрит фильм или читает книгу. Если это путешествие, то отправляется в путешествие.
6. Покупатель оценивает товар. Иногда покупатели возвращаются в магазин/на сайт, чтобы оценить товар.
7. Покупатель перепродает товар или избавляется от него иным способом. Взаимодействие покупателя с этим товаром завершилось. Он утилизирован, стерт или перепродан – в последнем случае, вероятно, товар снова повторит этот цикл взаимодействия.

Какие данные можно получить на каждом этапе, мы рассмотрим чуть позже. Но обратите внимание, что явная обратная связь в виде оценок поступает на этапе под номером 6 или даже позже. Это один из последних этапов взаимодействия. Поэтому, несмотря на то что люди в первую очередь говорят об оценках, необходимо получать данные задолго до того, как эти оценки появятся.

2.2.2. Что можно узнать по поведению обозревателя в магазине

Теперь подробнее поговорим о том, что происходит на этапах 1–3 на рис. 2.3. Обозреватель – это посетитель, который просматривает контент. Он может случайным образом просматривать множество различных товаров, но часто останавливается на контенте, который кажется ему интересным или подходящим. В обычном магазине такой обозреватель болтается по торговому залу без какой-либо видимой цели или четкого представления, куда идти. В какой-то степени покупатель присматривается к товарам и готовится к будущим покупкам.

Обозреватель

Обозреватель – это покупатель, который изучает контент. Как я уже сказал ранее, обозревателю нужно показать как можно больше товаров, и рекомендации должны отражать эту задачу. Если вам удалось установить, что посетитель является обозревателем, вы можете, держа в уме эту информацию, формировать такие рекомендации, которые соответствуют «обозревательскому» настроению человека.

Здесь нужно собирать данные о том, на каких объектах покупатель останавливается и что изучает. Также есть смысл отслеживать, к чему из увиденного обозреватель не проявил никакого интереса. Но можете ли вы быть уверены на 100 %, что просмотр страницы (товара) – это всегда хороший знак?

Просмотр страницы

Просмотр страницы на коммерческом сайте может означать многое. Это может быть сигналом о том, что посетитель (или обозреватель) заинтересовался. Но может быть и так, что человек не может найти нужную ему страницу или просто щелкает по всем ссылкам подряд без разбору. В последнем случае большое количество щелчков – это не хорошо. Потерявшийся пользователь проявляет себя большим количеством кликов и отсутствием конверсий.

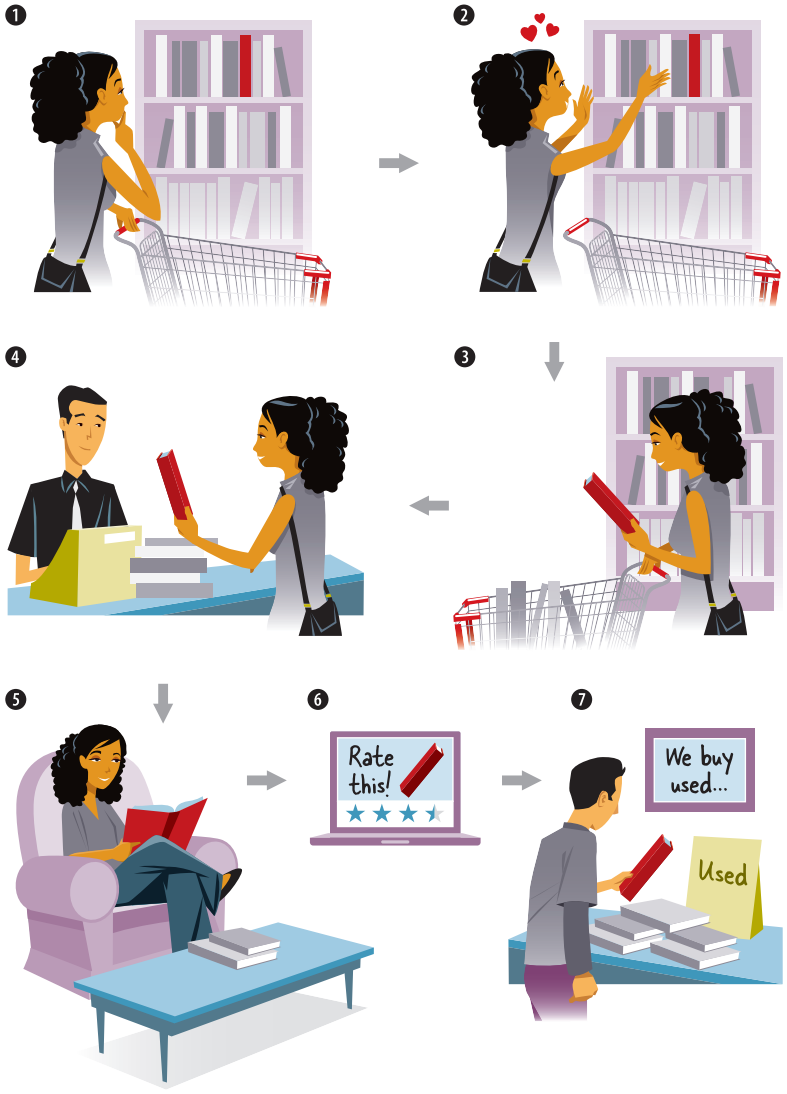


Рис. 2.3. Цикл взаимодействия покупателя с товаром

С другой стороны, чем лучше рекомендательная система, тем меньше страниц получат просмотры. Дело в том, что люди будут находить все, что ищут, благодаря ссылкам и товарам из рекомендаций, и им не нужно будет осматриваться.

Длительность просмотра страницы

Чтобы определить, что именно интересует посетителя, зашедшего на ваш сайт, можно посмотреть, сколько времени он провел на странице с тем или иным контентом. Но насколько точны эти данные? Точно, если исходить из того, что посетитель не занят ничем другим, и что следующее действие человека – это переход на новую страницу по ссылке, расположенной на текущей странице. В табл. 2.3 показан один из возможных вариантов трактовки того, что может обозначать длительность пребывания обозревателя на странице.

Таблица 2.3. Длительность просмотра страницы и возможное объяснение

Длительность просмотра страницы	Что это означает
Менее 5 секунд	Отсутствие интереса
Более 5 секунд	Интерес присутствует
Более 1 минуты	Посетитель очень заинтересовался
Более 5 минут	Возможно, пошел налить себе кофе
Более 10 минут	Отвлекли или ушел со страницы, не переходя по ссылке

Необходимо делать поправку на свою сферу деятельности, но в целом, как мне кажется, многие согласятся, что эти трактовки могут соответствовать действительности. Какие из этих данных нужно фиксировать? Все. Менее 5 с – не понравилось; от 5 с до 1 мин – возможно, заинтересован; 1–5 мин – возможно, пользователь думает, «это просто супер»; а 5 мин и больше – сложно понять. Все зависит от контента страницы. Это не точная наука.

Щелчки по ссылкам с подробностями

Помимо наблюдения за длительностью просмотра страницы, существуют и другие способы распознать интерес пользователя к тому или иному контенту. Разместите на странице простые контрольные действия, которые подскажут вам, чем занимается пользователь. Например, на интернет-сайтах часто присутствуют ссылки на подробную информацию, как показано на рис. 2.4. Это удобно посетителям, которые получают возможность быстро просмотреть контент или, в случае заинтересованности, развернуть описание. Еще вариант – пользователь может прокрутить страницу вниз, чтобы прочитать отзывы или технические характеристики. Если пользователь совершает одно из этих действий, это говорит о его заинтересованности.

Рис. 2.4. Щелчок по ссылке, разворачивающей описание, свидетельствует об интересе со стороны посетителя. На рисунке пример с сайта Amazon.co.uk

Ссылки на социальные сети

Также можно разместить кнопку перехода в социальные сети (рис. 2.5) для тех людей, которым что-нибудь понравилось настолько, что они решили поделиться своими впечатлениями с остальными. Контролировать происходящее на Facebook, в Twitter или в других социальных сетях не получится, но вы сможете зафиксировать сам факт того, что посетитель поделился информацией о том или ином контенте.

Рис. 2.5. Стандартный набор ссылок на соцсети

Сохранить на потом

Возможность сохранить что-то на потом, когда пользователь добавляет объекты в свой список, – это мощный ресурс. Если посетитель нашел для себя что-то интересное, очень уместно предоставить ему возможность сохранить эту вещь на потом (если он не готов купить это сразу же). Достаточно просто разместить ссылку на добавление страницы в закладки. А еще лучше создать список желаний, раздел **Избранное** или **Посмотреть позже** – в зависимости от типа контента. К другим признакам заинтересованности относятся скачивание брошюры, просмотр видео о каком-то элементе контента или подписка на новостную рассылку по определенной тематике.

Поисковые запросы

Посещение интернет-сайта может означать как то, что люди просто проявляют любопытство, так и то, что они ищут что-то конкретное. Если страница оформлена должным образом, большинство покупателей найдут, что им нужно, довольно быстро. По словам Netflix, каждый раз, когда человек пользуется поиском, свидетельствует о неудаче со стороны рекомендательной системы, поскольку говорит о том, что люди не нашли ничего интересного для себя в рекомендациях. Не знаю, могу ли я с этим согласиться, ведь нередко я пользуюсь поиском, когда кто-нибудь из знакомых посоветует фильм, непохожий на то, что я обычно смотрю. Так или иначе, поисковые слова лучше всего подскажут, что нужно посетителю.

На рис. 2.6 показано окно поиска сервиса Netflix. На сайте больше фильмов, но доступны к просмотру не все. Поэтому, если вы ищете «Чудо-женщину», вы увидите похожие названия, хотя самой «Чудо-женщины» в каталоге нет.

Даже если система не может дать пользователю тот контент, который он ищет, фиксировать подобные события все же следует. Если пользователь ищет фильм, вы узнаете, что ему интересен подобный контент. Зная это, рекомендательная система сможет предложить человеку что-то похожее.

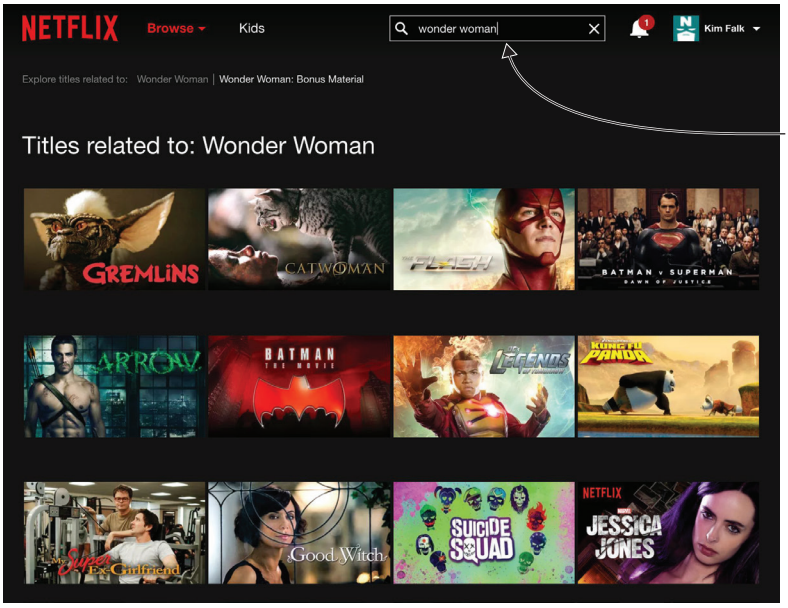


Рис. 2.6. Окно с результатами поиска «Чудо-женщины» на Netflix

Связь поисковых слов и дальнейших действий пользователя

Еще один момент, который может стать пищей для размышлений, связан с *поисковыми запросами* (то, что пользователь печатает в поисковом поле). Имеет смысл соотносить то, что пользователь искал, с тем, что он в итоге выбрал. Например, пользователь ищет «Звездные войны» и просматривает описание к «Космическому пирату Харлоку», который находится в одном списке с «Вавилоном нашей эры», и пользователь в итоге смотрит именно «Вавилон нашей эры». Возможно, есть смысл добавить «Вавилон нашей эры» в поисковую выдачу по запросу «Звездные войны».

2.2.3. Совершение покупки

Покупка чего-либо означает, что пользователь считает объект полезным или привлекательным. Или же покупает что-то в подарок. Нелегко определить, для кого предназначается покупка: для самого покупателя – и тогда это будет еще одним фактом, указывающим на предпочтения пользователя, или для кого-нибудь другого – и тогда эти данные нужно проигнорировать.

Понять, какие из покупок на самом деле являются подарками, а какие нет – это интересная задача. Объект, выбивающийся из общего ряда предыдущих предпочтений пользователя, может указывать либо на новую грань его интересов, либо быть подарком. В любом случае он будет белой вороной на фоне остальных данных.

В графическом виде такая белая ворона может быть представлена как одинокая точка, расположенная на большом расстоянии от основного скопления точек, как показано на рис. 2.7. Поскольку невозможно со стопроцентной точностью определить, что эти белые вороны означают (подарок или новый интерес), как правило, лучше не принимать их во внимание. Однако иногда это может быть первым сигналом о появлении нового круга интересов, дающего широкий простор для изучения.

Сам факт покупки свидетельствует о том, что объект был представлен наилучшим образом, но никак не указывает на то, понравился ли купленный товар покупателю. По крайней мере, это так, если человек покупает данный товар впервые. Можно возразить, что каждая покупка банана повышает оценку бананов. Первая покупка, возможно, непоказательна, но вторая уже может о чем-то говорить. Как бы там ни было, покупку можно расценивать как положительное событие.

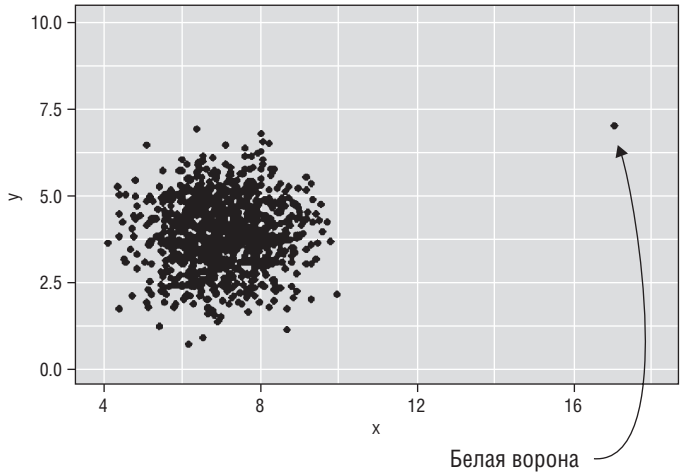


Рис. 2.7. Пример объекта, выбивающегося из общего ряда

2.2.4. Пользование товаром

Когда что-то покупается, магазин утрачивает контакт с товаром и возможность узнать, как его применяют, – если у вас не сайт, предлагающий стриминговые мультимедиа или стриминговый сервис.

Endomondo

Фильмы и музыка – это не весь контент, потребляемый в онлайн-режиме. Endomondo (**www.endomondo.com**) – это пример альтернативного интернет-сервиса. Это социальная сеть, посвященная фитнесу, где пользователи могут сохранять данные о своей физической активности с помощью специальных трекеров.

Endomondo отслеживает, насколько интенсивно человек пользуется теми или иными функциями, чтобы порекомендовать аналогичные сервисы или понять, какие аспекты необходимо доработать. Кстати, телефонные компании тоже анализируют то, каким образом люди пользуются телефонами. Они могут следить за нашими действиями совершенно пугающим образом. В следующем разделе мы поговорим, какую информацию можно получить благодаря стриминговому контенту.

Стриминговый контент

Когда дело касается стриминговых сервисов, музыки, фильмов и даже книг, любое действие пользователя можно расценивать как неявную оценку. Прослушивание песни говорит о том, что пользователю она нравится. Но можно копнуть еще глубже. Следующий список отражает возможные варианты взаимодействия пользователя с музыкальным контентом или фильмами:

1. *Запуск воспроизведения*. Пользователь заинтересован, это уже хорошо.
2. *Остановка воспроизведения*. О, подождите, возможно, пользователю было просто любопытно, но потом он решил, что это ужасно, и остано-

вил воспроизведение. Если песню остановили на первых 20 с (а фильм на первых 20 мин), это может быть дурным сигналом. Если остановили ближе к концу, вариантов больше.

3. *Продолжает воспроизведение*. Ладно, забываем все подразумеваемые плохие оценки, зарегистрированные системой. Возвращение к просмотру/прослушиванию контента после остановки может означать разное. Если пауза длилась менее 5 мин, вероятно, кто-то или что-то отвлекло пользователя, поэтому эту остановку учитывать не нужно. Но если пользователь возвращается к контенту спустя 24 ч – скорее всего, этот контент ему нравится.
4. *Перемотка*. Если пользователь пропускает какой-то фрагмент в середине, это, скорее всего, не самый хороший сигнал, но только при условии, что это первый раз, когда человек смотрит/слушает материал. Если, например, фильм смотрят уже по десятому разу, пропуск надоевших сцен ничего не говорит об отношении к фильму в целом. Технический редактор этой книги заметил, что, слушая музыку, он часто проматывает песню вперед, чтобы понять, стоит ли ее слушать. В случае с песнями контекста меньше, поэтому перемотка – это один из способов понять, понравится вам песня или нет. С фильмами ситуация иная.
5. *Воспроизведение от начала до конца*. У нас появился победитель! Это хороший знак. Пусть даже пользователь не поставит высокую оценку, но раз он дождался до самого конца, вероятно, он будет смотреть похожие фильмы в дальнейшем. (Фильм считается просмотренным до конца, если воспроизведение не было остановлено до того момента, пока не начались финальные титры.)
6. *Повторное воспроизведение*. Пересмотр контента может быть расценен как нечто положительное, если речь идет о фильме или музыке. Но, например, для сайта с обучающими видеороликами это может обозначать, что материал был слишком сложным.

Эти этапы применимы к большинству стриминговых материалов. Методы сбора фактов о том или ином продукте зависят от того, какой проигрыватель применяется.

Что касается Endomondo, который тоже относится к одной из разновидностей стриминговых сервисов, вышеописанные этапы взаимодействия не слишком показательны. Ведь в этом случае, если вы запустили Endomondo и указали, что начали пробежку (нажав кнопку запуска), но вскоре после этого нажали на паузу, это не значит, что вам не нравится приложение. Возможно, вам просто нужно привести себя в форму.

2.2.5. Оценки посетителей

Наконец, мы добрались до наиболее обсуждаемой темы – оценок. У сервиса Netflix есть девиз: «Чем больше вы ставите оценок, тем лучше ваши рекомендации».

Как вы увидите позже, это можно считать правдой с некоторыми оговорками. Большинство рекомендательных систем применяют оценки, но при этом оценки пользователей обычно сопоставляются с их поведением. Для этих систем оценки – это лишь опорная точка. Самое главное, что нужно понять, – это поведение пользователей.

Многие сайты позволяют пользователям давать отклик на контент, который они просмотрели, купили или применили. Так система может получить представление о том, что нравится пользователям, а следовательно, и о том, что предлагать им в будущем. Выставляя определенное количество звезд (или шляп, смайликов или чего бы то ни было еще), мы даем оценку, которая, по сути, представляет собой всего лишь число на цифровой шкале.

Amazon, как и большинство площадок, пытается с помощью подсказок сориентировать вас, что означает то или иное число звезд. На рис. 2.8 показан пример отзыва на книгу на сайте Amazon.

Рис. 2.8. При оценке любого контента на Amazon площадка подсказывает, что обозначает то или иное количество звезд

На Amazon, когда пользователь наводит курсор на звезды, появляется описание. В данном случае четыре звезды означают «мне понравилось». Помимо оценок, некоторые сайты, такие как TripAdvisor, подталкивают пользователей написать отзыв.

Считается, что оценка в пять звезд в сочетании с текстовым отзывом – это лучше, чем просто оценка в пять звезд, потому что написавший отзыв человек подошел к выставлению оценки более осмысленно. То же самое касается оценок с одной звездой. Но если человек сопровождает каждую свою оценку письменным отзывом, то это ничего не значит. Может ли покупка чего бы то ни было без выставления оценки говорить что-либо о ваших предпочтениях?

Ощущение контроля

Когда в магазинах только появились сухие смеси для выпечки кексов (рис. 2.9), они с треском провалились. Казалось, что этот товар идеально подходит для занятых покупателей: единственное, что требовалось, – это просто

развести смесь водой. Было проведено исследование, в результате которого выяснилось, что проблема заключалась не в том, что процесс приготовления был прост, а в том, что он был *слишком* прост. Выпекание кекса – это творческий процесс, а полуфабрикаты свели его к набору примитивных действий, лишив покупателей контроля над ситуацией. Производители на это ответили: «Хорошо, пусть они также добавляют яйца». Это удешевило производство и наделило покупателей большей свободой. Когда на прилавки вернулись обновленные смеси для выпечки кексов, в которые необходимо было добавлять воду и яйца, они стали пользоваться огромной популярностью.

Рис. 2.9. Смесь для приготовления шоколадного кекса. В первом ее варианте покупателям нужно было просто развести ее водой. Но она не имела успеха, пока состав не изменили таким образом, чтобы в смесь требовалось добавить не только воду, но и яйца

Многие сайты предоставляют посетителям возможность указывать свои предпочтения с той же самой целью: наделить пользователя ощущением контроля над тем, какие предпочтения приписывает ему система. По наблюдениям Netflix, многие люди указывают, что им нравится документальное и зарубежное кино, а сами смотрят американские ситкомы. Что в этом случае должна им предлагать рекомендательная система Netflix: контент, заставляющий их стыдиться своего бесцельного времяпрепровождения, или контент, который им действительно хочется посмотреть?

Вот почему так трудно оценить эффективность рекомендательной системы, опираясь на данные с рейтингами пользователей. Эти данные могут подсказать вам, насколько точны прогнозы рекомендательной системы, но не помогут понять, привлечет ли система дополнительных пользователей.

Сохранение оценок

Когда пользователь ставит продукту оценку, это событие тоже нужно сохранять, как и любое другое. Возможно, стоит выполнять сохранение непосредственно в базе данных контента, чтобы можно было вывести среднюю оценку контента при его показе пользователю.

«Плохие» оценки пользователей

Дело становится немного сложнее, если вы, как потребитель, хотите дать плохую оценку, поскольку для этого нужно поставить хотя бы одну звезду. Ноль звезд означает, что контент вообще не был оценен. Если вам ужасно не нравится контент – вы не хотите ставить ему никаких звезд, даже одну. В некотором смысле отсутствие оценки лучше, чем плохая оценка или отметка «ужасный товар», которую на Amazon можно поставить товару с одной звездой, как показано на рис. 2.10.

Если вам что-то не нравится, то, возможно, вы и оценку ставить не захотите. Но если что-то раздражает вас по-настоящему сильно, вы можете захотеть выразить разочарование, например в виде отрицательного отзыва.

Оценка, показанная на рис. 2.10, не моя. Я-то как раз определенно рекомендую книгу «Deep Learning» (Издательство MIT, 2016), если у вас есть базовые знания машинного обучения. В противном случае вам будет лучше начать с «Grokking Deep Learning» Эндрю Траска, (Manning, 2016).

Рис. 2.10. На Amazon одна звезда означает «ужасный товар». Но эта книга мне нравится, и если вы хотите изучать машинное обучение, то эта книга – то, что надо

Голосование

Многим сайтам удалось сформировать сообщество пользователей, которые оценивали продукты путем голосования. К примеру, у TripAdvisor есть сервис оценки гостиниц и ресторанов. Другим примером является Hacker News (**news.ycombinator.com**), где пользователи занимаются добавлением контента, а именно ссылок на статьи и сообщений о «чем-нибудь, что понравится хакерам». При добавлении контента его можно поднять *своим голосом*. Чем их больше, тем выше контент поднимается на странице (тут все проще некуда, мы рассмотрим этот алгоритм позже). Сайты, на которых используется голосование, называются *системами репутации*.

2.2.6. Знакомство с клиентами по (старому) методу Netflix

Если контент на странице почти полностью состоит из предложений, важно собрать как можно больше данных о вкусах потребителя. Если Netflix считает, что ваши вкусы чересчур отличаются от рекомендаций, найти что-нибудь посмотреть может быть трудно. Этим пользуются люди, которые утверждают, что рекомендательные системы манипулируют мнением. Они считают, что, если система не дает равные возможности всем элементам контента, она манипулирует мнением. Я понимаю этот аргумент, но с мнением не согласен.

Как-то раз Netflix дал пользователю возможность помочь платформе в создании профиля вкуса[1]. Эта функция больше не доступна, но когда-то она была в меню профиля вкуса, как показано на рис. 2.11. Эта функция позволяет пользователю оценивать сериалы и фильмы и выбирать жанры на основе того, как часто пользователь смотрел, например, «Адреналин». Netflix также может проверить, соответствуют ли оценки пользователя его текущим вкусам.

В Netflix использовался ручной ввод информации о вкусах, а затем платформа составляла предложения. Подход, в котором пользователь помогает составлять свой профиль вкуса, часто используется с целью дать системе возможность составлять предложения для новых пользователей.

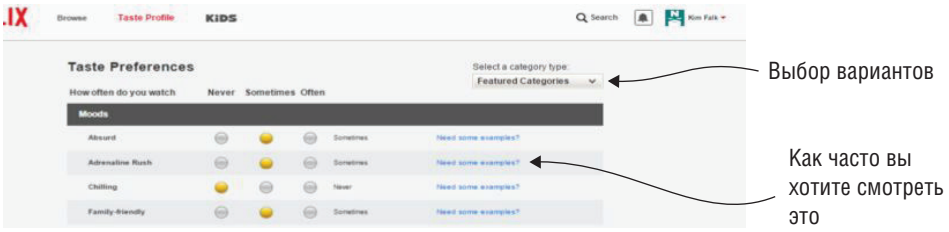


Рис. 2.11. Настройки вкуса Netflix 2015 года. Сейчас функция уже недоступна

2.3. Идентификация пользователей

Сбор данных о пользователях работает только в том случае, если у вас есть способ однозначно идентифицировать их. Лучший способ сделать это – ввести авторизацию на странице, чтобы вы точно знали, кто к вам зашел. Другим способом является использование куки.

Обычно сайты начинают с загрузки файлов куки и собирают всю информацию в эти файлы куки. Если после этого пользователь обеспечивает идентификацию путем входа в систему или создания профиля, вся информация из куки передается в этот профиль. Будьте осторожны с куки, потому что компьютер может использоваться несколькими людьми, особенно в общественном месте. Сохранение данных в куки в течение нескольких сеансов может ввести систему в заблуждение.

[1] Подробнее: **help.netflix.com/en/node/10421**.

Если у вас нет авторизации пользователей, возникает также проблема использования нескольких устройств. Это означает, что, даже если вы определили пользователя на одном устройстве, ваша система не сможет узнать его на других устройствах. Есть сервисы, которые могут помочь вам в этом, но результат не стопроцентный. Стоит все-таки ввести регистрацию и авторизацию пользователей, если это возможно. Само собой разумеется, что персональные рекомендации могут работать только в случае, если вы опознали пользователя.

2.4. Получение данных о посетителях из других источников

Ваш сайт уникален, и собирать следует только те данные, которые лучше всего подходят для анализа поведения ваших клиентов. Но что, если бы вы могли пойти на хитрость и собрать данные из другого места?

Для начала можно использовать соцсети, и, если вам повезет, посетителю вашего сайта понравится что-то в вашем каталоге. В зависимости от вашего контента для сбора данных можно использовать Facebook, LinkedIn или другие подобные сайты. Многие думают, что авторизация через Facebook – это хорошо, но если речь идет не о фильмах или книгах, то какой толк от данных Facebook? Однако большинство сайтов сможет извлечь пользу даже из простых данных возраста и места жительства посетителя.

Могут быть и другие способы получить знания. Рекомендательная система пенсионных планов может извлечь пользу из знания о том, читает ли клиент книги о финансах. Это было бы свидетельством того, что клиент интересуется фондовыми рынками и, вероятно, будет более заинтересован в пенсионных планах, позволяющих клиенту иметь больше контроля над портфелем.

Еще стоящая рассмотрения вещь заключается в том, что многие алгоритмы расчета рекомендаций основаны на подобии пользователей. Если система имеет некоторые данные о нескольких пользователях, даже если данные могут не иметь отношения к контенту вашего сайта, это позволит системе найти похожих пользователей, что затем может быть использовано для создания рекомендаций. Я уверен, что сайт автомобильного дилера может случайно обнаружить, что владельцам внедорожников нравится определенная резина, а сайт с косметикой может рекомендовать что-то в зависимости от возраста и пола. Люди, которые работают в сфере ИТ, вероятно, любят смартфоны, а водители – солнечные очки.

2.5. Сборщик данных

Теперь рассмотрим сборщик данных, реализованный на сайте MovieGEEKs. Мы изучим его существенные части, а затем оставим код на более подробное изучение заинтересованному читателю.

В связи с особенностями производительности и надежности вашего сайта лучше не добавлять сбор данных в имеющееся (веб-) приложение. Вместо этого добавьте его в параллельную структуру, которая подходит для вашей цели. Это позволит переместить сборщик данных на другой сервер, если пользователи вдруг загрузят ваш рекомендатор до предела и нагрузка на вашем сайте приблизится к своему пределу.

Наш сборщик данных разделен на две логические части.

На стороне сервера. В нашем примере часть на стороне сервера построена с использованием веб-API Django, которое работает в качестве конечной точки и к которому может обращаться все, с чем контактирует пользователь. Обычно это веб-страницы, но также это может быть приложение на телефоне или любом устройстве, подключенном к интернету и собирающем данные о соответствующих событиях. Когда сервер получает уведомление о том, что произошло событие, ему требуется лишь сохранить его. Веб-API представляет собой HTTP-адрес, настроенный для получения такого рода сообщений, и может быть реализован в любом фреймворке.

На стороне клиента. Клиентской части в традиционном смысле этого слова здесь нет, потому что нет запроса к веб-страницам. Клиентская часть будет состоять из простой функции на JavaScript, которая передает данные сборщику на сервере.

При наличии сборщика вы сможете начать сбор данных как на MovieGEEKs, так и на своем сайте. Чтобы связать сборщик с остальной частью архитектуры MovieGEEKs, на рис. 2.12 представлены элементы сборщика в схеме архитектуры из главы 1.

Вы можете подумать, почему бы не использовать средства Django и забыть о сторонних приложениях? Но поразмышляйте вот о чем:

- журнал Django работает при выполнении кода на стороне сервера, так что у вас не будет способа сохранения данных о поведении пользователя (кроме как поместить его в своего роде сессию и сохранить в следующем запросе). Все события вроде наведения курсора и прокрутки будут потеряны;
- сборщик данных позволяет получать данные отовсюду, а не только с веб-сайта. В будущем ценными данными будет что угодно, вплоть до информации о том, что вы зашли на сайт из конкретного магазина.

Поскольку собранные данные имеют простой формат, лучше сохранять их в разделенный запятыми файл (CSV). В этом случае будет легче перемещать данные, если система будет удаленно расположена.

Файл CSV можно отправить в сервис, способный обрабатывать данные со скоростью, с которой справится система CSV. Таким образом, у вас всегда есть буфер, позволяющий убедиться, что система не перегружена. Поскольку запрашивать CSV-файл неудобно, а запросы нам точно понадобятся, вместо этого мы будем использовать базу данных.

Рис. 2.12. Архитектура сборщика и регистратора данных MovieGEEKs

2.5.1. Создание файлов проекта

Скачать и запустить сайт довольно легко. Вам нужно загрузить или клонировать репозиторий GitHub по адресу **mng.bz/AU9X**. После завершения загрузки следуйте инструкциям, приведенным в файле *readme.md*.

2.5.2. Модель данных

Очень важно собрать данные о большинстве видов взаимодействий. В предыдущем разделе мы говорили, что вам нужно всего три вещи: идентификатор пользователя, идентификатор контента и тип события. На самом деле нужно еще несколько вещей, приведенных в модели на рис. 2.13.

Параметр `session_id` однозначно идентифицирует каждую сессию. Позже вы сможете добавить в модель, например, тип устройства или контекст. Но пока что можно начать с этого.

Logger	
date:	datetime
user_id:	varchar(64)
content_id:	varchar(16)
event:	varchar(200)
session_id:	integer

Рис. 2.13. Информационная модель регистратора данных

Покопавшись во внутренностях регистратора, в который вы всегда можете заглянуть, вы обнаружите, что это веб-API, который всегда открыт для одного типа запросов, содержащих данные, приведенные на рис. 2.13. Мы подробнее разберем запросы немного позже, когда будем говорить о том, как взломать сайт MovieGEEKs. Но сначала давайте посмотрим, что нужно сделать на стороне клиента.

О своевременности

Время записи – это штука с подвохом, потому что нужно учитывать часовые пояса. Использование данных локальных временных зон означает, что порядок событий может перепутаться из-за разницы часовых поясов. В то же время запись местного времени означает, что можно будет правильно обрабатывать такие фразы, как «во второй половине дня», вместо того чтобы рассматривать разные промежутки времени для каждого часового пояса.

2.5.3. Сборщик данных на стороне клиента

Данные можно собирать в виде событий из всего, что взаимодействует с пользователем, от телефонного приложения до устройства, которое вы положили в обувь во время пробежки.

Регистратор событий является простой функцией JavaScript, которая вызывает сборщик событий. Он не делает ничего в случае ошибки, потому что конечные пользователи с этим ничего сделать не могут. В реальной работе стоит отслеживать факт записи данных, но, видимо, это нужно сделать не здесь.

Сборщик должен находиться в той части проекта, где вы хотите собирать данные. Вы найдете файл под названием *collector.js* в папке */moviegeek/static/js*, содержащей функцию, показанную в листинге 2.1, который создает вызов AJAX к сборщику.

ЛИСТИНГ 2.1. Создание вызова AJAX к сборщику: */moviegeek/static/js/collector.js*

```
function add_impression(user_id, event_type, content_id,
                        session_id, csrf_token)
{
  $.ajax(
            type: 'POST',
            url: '/collect/log/',
            data: {
                "csrfmiddlewaretoken": csrf_token,
                "event_type": event_type,
                "user_id": user_id,
                "content_id": content_id,
                "session_id": session_id
            },
  fail: function(){
                console.log('log failed(' + event_type + ')')
                }
            })
};
```

Сообщение вызыв ется методом POST, т к к к вы доб вляете что-то в б зу д нных

Вызов AJAX

Токен CSRF, который позволяет с йту вызыв ть с йт с другого домен

Отобр жение трех в жных элементов д нных

Отобр жение уник льного идентифик тор сессии

В случ е неуспех информ ция з писыв ется в консоль отл дки бр узер . Пользов телю ничего не выводится

Вызовы функции в листинге 2.1 в конечном итоге будут получены методом журнала в */moviegeek/collector/view.py* – см. листинг.

ЛИСТИНГ 2.2. Прием вызова из листинга 2.1: */moviegeek/collector/view.py*

```
@ensure_csrf_cookie
def log(request):

    if request.method == 'POST':
        date = request.GET.get('date', datetime.datetime.now())

        user_id = request.POST['user_id']
        content_id = request.POST['content_id']
        event = request.POST['event_type']
        session_id = request.POST['session_id']

        l = Log(
            created=date,
            user_id=user_id,
            content_id=str(content_id),
            event=event,
            session_id=str(session_id))
        l.save()
    else:
        HttpResponse('log only works with POST')


    return HttpResponse('ok')
```

Этот метод берет только сообщения тип POST

Созд ет метку времени, доб вляемую в з пись

Сохр няет з пись журн л в б зу д нных

Хороший ответ, д же если сообщение не тип POST

2.5.4. Интеграция сборщика в MovieGEEKs

В приложении MovieGEEKs есть примеры, приведенные в табл. 2.4. Примеры аналогичны тем, что перечислены в табл. 2.1, где приведены варианты поведения пользователя на сайте Netflix. Я добавил сюда таблицу со столбцом данных о событиях, которые будут собраны.

Таблица 2.4. Точки данных MovieGEEKs

Событие	Значение	Точка данных
Щелчок по жанру, например драмы	Пользователь заинтересован в жанре (в данном случае драмы)	(`Kimfalk`, `drama`, `genreView`)
Наведение мыши на фильм «История игрушек» (запрашивает обзор контент)	Пользователь заинтересован в фильме	(`Kimfalk`, `ToyStory`, `details`)
Щелчок по фильму (запрос подробностей о фильме)	Пользователь еще больше заинтересован в фильме	(`Kimfalk`, `ToyStory`, `moreDetails`)
Нажатие кнопки **Отложить на потом**	Пользователь намерен посмотреть фильм	(`Kimfalk`, `ToyStory`, `addToList`)
Нажатие кнопки **Купить**	Пользователь смотрит фильм	(`Kimfalk`, `ToyStory`, `playStart`)

Регистрация событий жанров

Приведенные ниже события были реализованы в файле *templates/moviegeek/base.html*. Первое событие после входа – это щелчок пользователя по жанру. Жанры перечислены на левой стороне экрана, как показано на рис. 2.14.

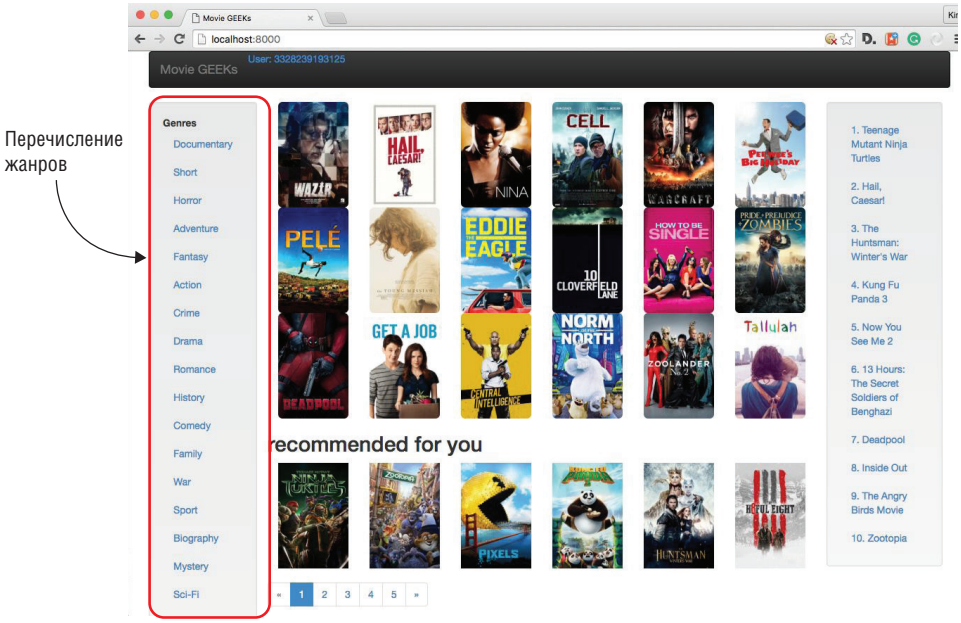


Рис. 2.14. Домашняя страница MovieGEEKs

Для регистрации щелчков по жанру к каждой ссылке добавляется атрибут `onclick`. Когда активный пользователь щелкает по жанру, он отправляет сборщику HTTP запрос POST. Мы не будем углубляться в код, потому что мы уже рассматривали подобное. Вы можете посмотреть на рис. 2.15, чтобы начать понимать, что происходит, когда пользователь щелкает по жанру.

1. Пользователь щелкает по жанру.
2. Выполняется событие `onclick`, который вызывает функцию JavaScript из листинга 2.1.
3. Выполняется функция `add_impression`.
4. Веб-сервер получает запрос HTTP, который делегирует его на сайт MovieGEEKs.
5. Сайт MovieGEEKs выполняет поиск в списке URL и делегирует все, что имеет URL/collector/, приложению коллектора.
6. Приложение коллектора соотносит log/ с методом просмотра.
7. Метод просмотра создает объект журнала.
8. Используя систему Django ORM, в базу данных выполняется запись[1].

[1] Подробнее: **help.netflix.com/en/node/10421**.

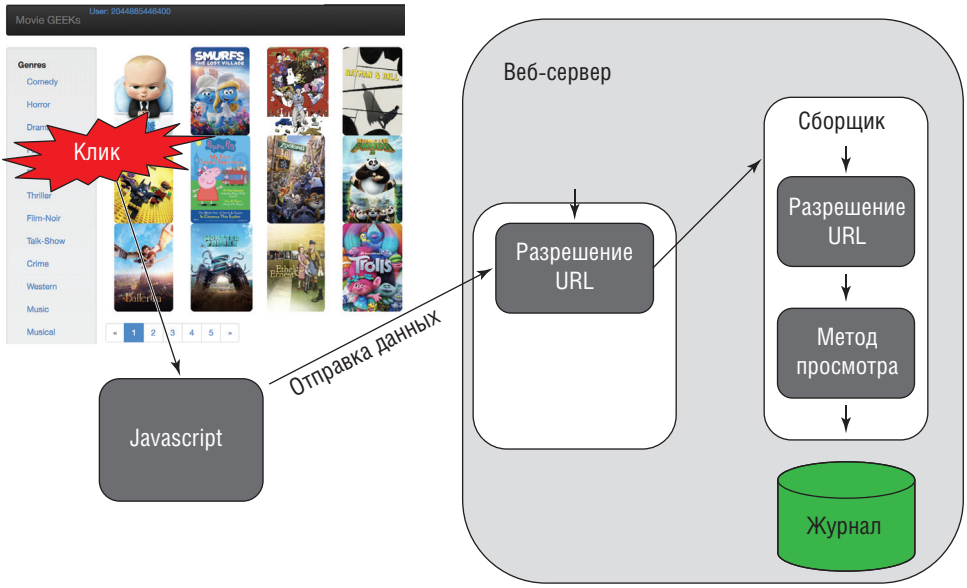


Рис. 2.15. Что происходит, когда пользователь щелкает по жанру

Регистрация наведения курсора

Когда пользователь нажимает на фильм, появляется всплывающая подсказка. На рис. 2.16 показано, как она выглядит. Щелчок пользователя указывает на то, что пользователь может быть заинтересован в фильме, и, следовательно, это событие записывается в журнал. Это делается путем добавления обработчика событий, который вызывает сборщик данных при каждом отображении подсказки.

Регистрация просмотра подробностей

Пользователь, прочитавший информацию в подсказке, может заинтересоваться и нажать кнопку More Details. Это тоже стоит отметить, поскольку это показывает дальнейший интерес к фильму.

Регистрация «сохранения на потом»

Вместо того чтобы нажать кнопку More Details, пользователь может добавить элемент в список. Это важное событие, так как оно означает, что пользователь планирует купить фильм позже. Хорошо иметь такую функцию. Это ссылка на просмотр подробностей, которая записывает событие «сохранения на потом». Вы также можете записывать другие события, но для этого примера остановимся на достигнутом.

2.6. Какие пользователи есть в системе, и как их моделировать

Прежде чем двигаться дальше, нужно немного подумать о пользователях. Мы уже говорили о поведении пользователей, но также может быть полезно учитывать, что волнует пользователей. Иными словами, нужно «знать своего клиента».

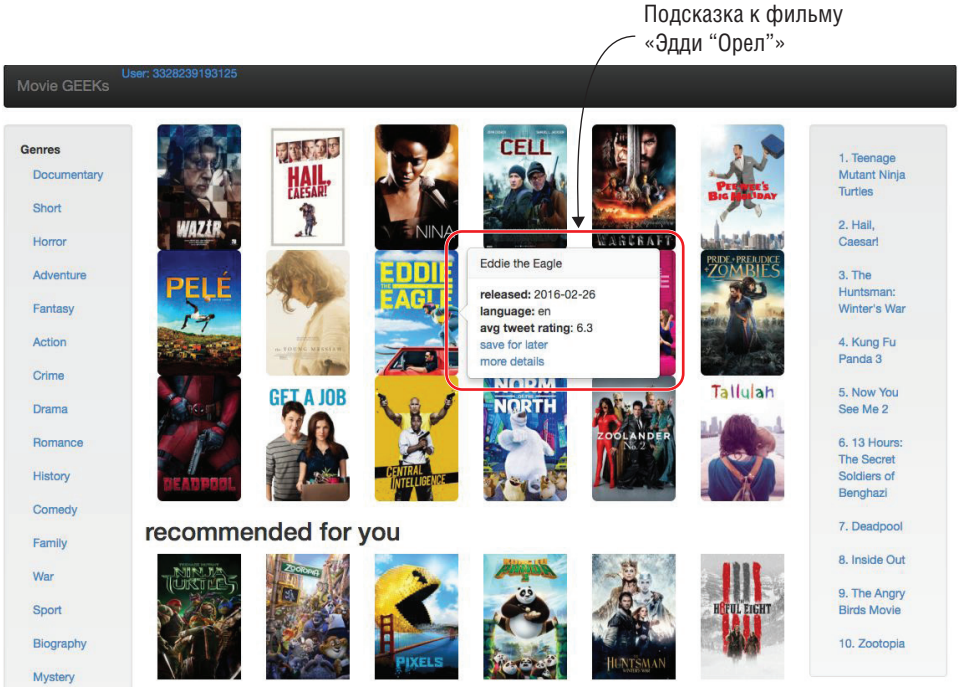


Рис. 2.16. Страница MovieGEEKs с подсказкой

Вам нужна модель пользователя, которую можно превратить в табличную базу данных и использовать в качестве дополнительных данных.

Что нам стоит знать о пользователе? Ответ, как всегда, «зависит от ситуации». Я ненавижу этот ответ, потому что это один из самых бесполезных ответов, наряду с «да, но не совсем» и «да, но не сейчас». Если предположить, что вы со мной согласны, то давайте представим, что ответа не существует, и рассмотрим различные варианты того, какие знания нам нужны. Тут можно также сказать, «мы хотим знать все». В теории для рекомендаций так и нужно.

Если вы разрабатываете рекомендательную систему вроде Jobsite (**www.jobsite.co.uk**), то стоит собрать информацию, например о текущем местоположении, образовании, опыте работы и т. д. Если вы делаете пенсионный сайт, вам, вероятно, потребуется тот же набор, а также данные о состоянии здоровья (например, как часто пользователь посещает больницу и какие лекарства при-

нимает). Для магазина книг могут также потребоваться все вещи, упомянутые выше, потому что все они указывают на то, какие книги могут заинтересовать пользователя. Но на большинстве книжных сайтов используются данные вкуса и покупательские привычки.

Давайте скажем «все, что есть» вместо «зависит от ситуации». Допустим, в вашей системе есть пользователь Петро (рис. 2.17). Какую информацию о Петро нужно сохранить? Если у вас есть информация, показанная на рис. 2.17, что из этого вы сохраните в базе данных? Проблем с памятью в наши дни особо нет, так почему бы не сохранить все? В будущих главах мы рассмотрим, что с этим можно сделать.

Рис. 2.17. Новый пользователь по имени Петро

В идеале вам нужен список пар ключ/значение, связанный с идентификатором пользователя, адресом электронной почты, и, возможно, другой дополнительной информацией. Но помните, что вы делаете рекомендательную систему, поэтому можно также сохранить адрес доставки и тому подобные данные. Но для целей сайта MovieGEEKs это не так важно. Эта информация дает модель данных, показанную на рис. 2.18.

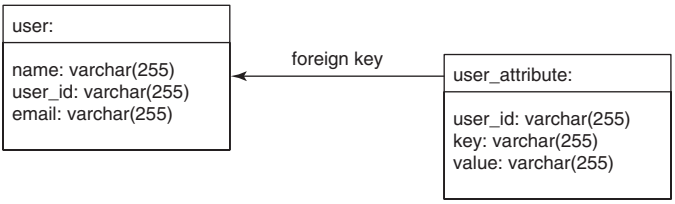


Рис. 2.18. Общая модель данных

Есть несколько вещей, к которым следует стремиться. Нам нужны:

- гибкость, чтобы сохранить все это;
- простота, чтобы сделать код читаемым.

Однако эти цели тянут повозку в противоположные стороны. Поэтому можно сделать менее гибкую, но более простую в использовании реализацию (где-то тут есть золотая середина). Вы можете создать таблицу, содержащую большинство предыдущих атрибутов, а также еще одну для каких-либо дополнительных данных, которые могут понадобиться позже. Ваша модель данных для пользователя может выглядеть, как показано на рис. 2.19.

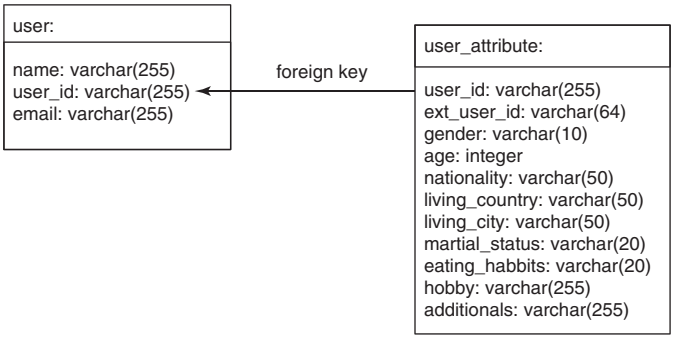


Рис. 2.19. Модель данных пользователя

Возвращаясь к нашему пользователю Петро, мы сохраним данные, показанные на рис. 2.20. Такая модель данных будет не слишком гибкой, но пока нам не нужна гибкая модель данных, а нужна модель попроще.

Если у вас есть постоянные посетители, в вашем журнале, возможно, уже имеется информация о них, и это здорово. Тем не менее всегда хорошо иметь информацию о пользователе, чтобы проверить, как работают ваши алгоритмы. В главе 3 мы поговорим о пользователях и автоматической генерации данных о них.

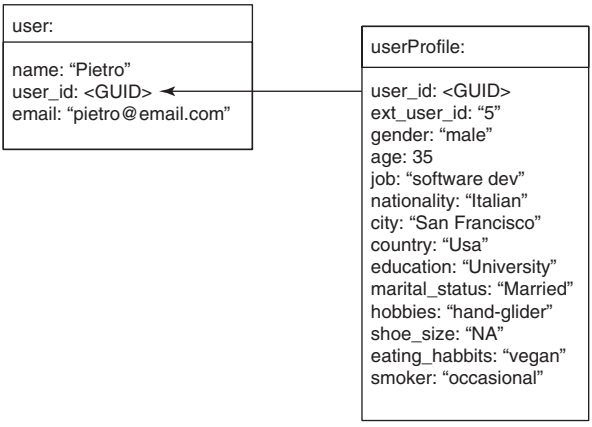


Рис. 2.20. Данные о пользователе Петро

Резюме

- Поведение пользователей записывается в журнал с помощью веб-API. Это, возможно, будет реализовано в другом веб-приложении, чтобы убедиться, что эта нагрузка не заставит сайт проседать в производительности, если пользователь запускает много событий.
- Подключите сборщик к веб-сайту, прикрепив скрипт вызова ко всем событиям, происходящим на сайте.
- Хорошие данные содержат информацию о системе вкусов пользователя. В идеале нужно записывать все события, потому что они могут оказаться полезными позже.
- Неявные оценки выводятся из событий, инициируемых пользователем, а явные ставятся пользователями напрямую.
- Неявные оценки, как правило, более надежны, но только если вы понимаете, какой смысл несет каждое событие.
- Явные оценки не всегда надежны, потому что они могут быть неточными в связи с влиянием социальных факторов.

Глава 3

Мониторинг состояния системы

Это короткая глава, но в ней есть важная информация:

- начнем с того, что все приложения, управляемые данными, должны начинаться с аналитики;
- я постараюсь убедить вас в том, насколько аналитика важна, и мы рассмотрим, как реализовать панель аналитики;
- я познакомлю вас с архетипами пользователей и расскажу, почему они полезны;
- с помощью архетипов мы изучим различные способы представить вкусы пользователей.

В предыдущих главах вы узнали, какой результат производит такая рекомендательная система и что вы можете узнать от пользователей, заходящих на ваш сайт. Здесь вы должны понимать, какого результата хотите достичь и какие данные вам для этого нужны. Теперь не хватает только двух шагов, показанных на рис. 3.1.

Для того чтобы понять эти два шага, вам нужно найти способ выяснить, чего хотят ваши пользователи. Взяв данные журнала, собранные по методу прошлой главы, вам нужно найти способ превратить данные каждого пользователя в информацию об их вкусах.

Рис. 3.1. Процедура от данных рекомендаций. Вы начинаете с данных, которые вы собираете, пока пользователи используют веб-сайт. С этим вы сможете начать понимать вкусы пользователя, которые будут играть роль входных данных для рекомендательной системы

Для этой цели можно использовать скрипты, которые автоматически генерируют взаимодействия, дающие вам нужные данные. Затем мы изучим основы аналитики, которую можно будет один раз настроить и постоянно просматривать.

Я всегда считал аналитику чем-то вроде непрерывного анализа данных. Она дает вам информацию о ваших данных, но это лишь малая часть происходящего. Однако в нашем случае этого может оказаться достаточно, как мы увидим в этой главе. Мы говорим об аналитике в книге о рекомендательных системах по той причине, что вам нужен способ понять происходящее и измерить результат вашей работы. Создание рекомендательных систем – это настолько весело, что вам может не быть до этого дела, но люди, которые платят деньги, хотят знать, если от рекомендатора не будет никакого толку.

Вы начнете свое путешествие в аналитику с изучения теоретической части того, что хотите показать. Затем, когда что-то начнет проясняться, рассмотрим это на примере сайта MovieGEEKs.

3.1. Почему панель аналитики – это круто

Когда я зарабатывал свою степень в области информатики, мы забавлялись над студентами, занимающимися визуализацией, называя их «цирковыми программистами»[1]. После университета я никогда не был большим фанатом свистелок и красивых графических интерфейсов. Я предпочитаю безмятежную красоту и порядок данных в таблице.

Занявшись большими данными и анализом данных, я увидел недочеты в моем пути и понял важность хорошей визуализации данных. Поработав с большими наборами данных в течение многих лет, я уверовал, что запуск приложения данных без их визуализации подобен вождению с закрытыми глазами: «Сначала великолепно, но пока вы поймете, что что-то не так, будет уже слишком поздно».

3.1.1. Ответ на вопрос «Как там дела у сайта?»

Давайте начнем с вопроса: «Как там дела у нашего сайта?» Как бы вы ответили на него? Что тут является критерием качества? Вырученные деньги? Количество посетителей? Среднее время отклика? Или что-то совсем другое?

Что вы хотите улучшить, добавляя на сайт рекомендательную систему? Первым результатом будет то, что ваши коллеги станут гораздо счастливее, так как им выпал шанс поработать с рекомендательными системами, что само по себе счастье. Но что это изменит? В этом и состоит вопрос данной главы. Панель визуализации данных имеет первостепенное значение для понимания того, «как дела» у сайта, как вы можете использовать собранные данные и удалось ли что-то улучшить.

Нам нужен способ оценивать текущую эффективность. Это настолько важно, что мне даже кажется, что вообще нельзя и пытаться реализовать рекомендательную систему, не имея панели аналитики. Я определенно рекомендую вам сперва создать аналитическую часть вашего сайта и уже потом добавлять рекомендательную систему.

[1] На датском это звучит слегка иначе, чуть менее обидно.

В следующих разделах мы рассмотрим реализацию панели аналитики на MovieGEEKs, так что вы сможете увидеть пример того, как отследить поток событий и поведение посетителей. Немного остановимся на наших целях.

Панели аналитики

Обычно компании стесняются рассказывать миру, как они отслеживают поведение или мониторят производительность, в основном потому что это может сыграть им во вред и дать хакерам информацию о слабых сторонах сайта. Кроме того, если ваши пользователи будут знать слишком много интимных подробностей алгоритма вашей рекомендательной системы, поведение пользователей может стать менее спонтанным. Это может вызвать перекосы в результатах или даже заставить пользователей искусственно подталкивать рекомендации в каком-то направлении.

В наше время, когда все идет к повышению персонализации, владельцы сайтов должны знать своих пользователей и их активность на сайте, чтобы включить управляемые данными решения и быстро реагировать на изменения. Визуализация данных поможет вам найти «узкие места». В этой главе мы рассмотрим лишь самые азы, и для глубокого понимания нужно будет изучить тему получше!

Панель аналитики MovieGEEKs

Панель аналитики MovieGEEKs показана на рис. 3.2. Хорошо бы знать об этих вещах, прежде чем приступать к реализации рекомендательной системы.

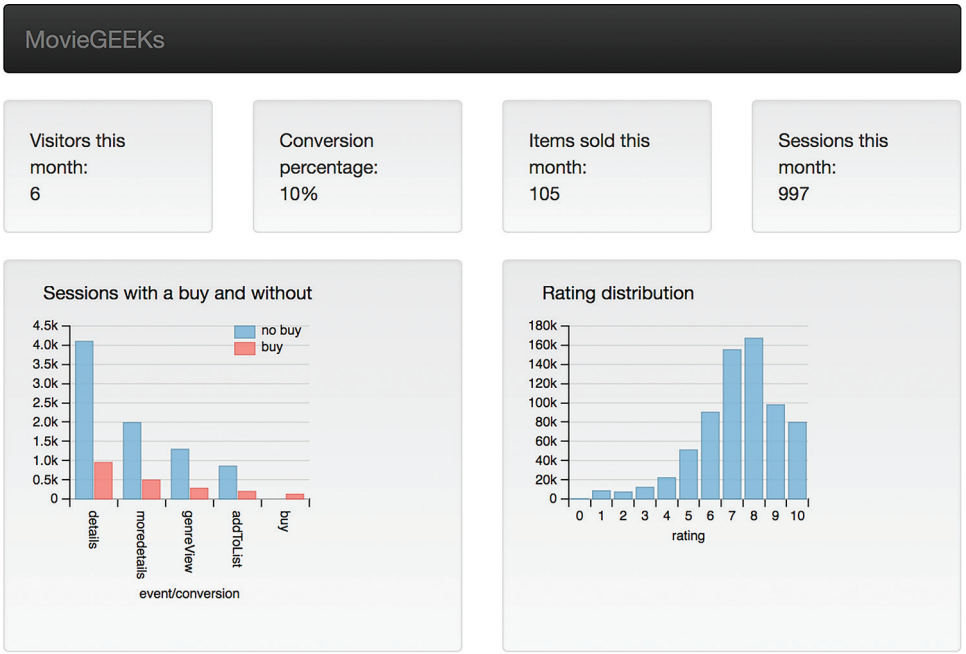


Рис. 3.2. Панель аналитики MovieGEEKs

3.2. Реализация аналитики

Если веб-сайт измеряет время отклика, это будет важным фактором в успехе сайта. Это обсуждалось в множестве книг. Здесь нас интересует деловая часть веб-сайта, потому что именно в ней мы узнаем, хорошие ли получаются рекомендации.

Дисклеймер

Неважно, насколько хорош рекомендатор. Его качество зависит от контента. Имея хороший контент, мы не сможем в мясном магазине рекомендовать что-либо вегетарианцу.

3.2.1. Веб-аналитика

То, что мы собираемся сделать, часто называется *веб-аналитикой*. Веб-аналитика подразделяется на две категории: вне сайта и на сайте. *Аналитика вне сайта* сосредоточена на возможности, видимости и голосах:

- *возможность* показывает, насколько велик потенциал сайта с точки зрения общего количества посетителей;
- *видимость* показывает, насколько легко можно найти сайт;
- *голоса* показывают, сколько людей говорит о сайте.

Эта глава посвящена аналитике на сайте, которая касается поведения пользователей на сайте (рис. 3.3). В этой категории основное внимание уделяется конверсии, драйверам и ключевым показателям эффективности (KPI), которые описаны в следующем разделе.

Рис. 3.3. ОПД в верхней части приложения аналитика

3.2.2. Базовые статистические данные

Анализ данных не так увлекателен, как поиск сокровищ в духе «Индианы Джонса», особенно если ваши данные похожи на то, что описано выше. В этом случае вам нужно будет реализовать визуализацию данных, которая позволит анализировать собранные данные. Это часто называют *статистическими данными*.

В верхней строке на панели показаны важные показатели текущего состояния KPI вашего сайта. В роли KPI может выступать что угодно, что позволяет оценить успех вашего сайта. Что например?

Во-первых, нам важно количество посетителей (так как в интернете принцип «построй, и они придут сами» не работает), поэтому это и будет первым показателем. Следующим идет уровень конверсии, о котором более подробно мы поговорим далее в этом разделе. Затем идет число элементов контента, проданных в этом месяце. Денежные показатели, такие как общий доход, также могут быть важны, но не в наших примерах. Но нам, безусловно, стоит отметить, что прибыль от различных элементов отличается, поэтому критерием успеха также может быть то, что вы продаете больше прибыльных элементов.

Числа на панели показывают статистику за предыдущий месяц, но данные также могут высчитываться ежедневно или еженедельно, или даже ежечасно, если у вас есть достаточно мощности и времени, чтобы анализировать все это. Вы также можете использовать скользящее окно, позволяющее вывести статистику в настраиваемый промежуток времени. Не столь важно, что вы делаете, пока это соответствует вашим целям.

Еще одна вещь, которую стоит учитывать при создании панели аналитики, заключается в том, что, если вы хотите проследить за развитием сайта, можно, например, нарисовать график, который показывает значения ключевых показателей эффективности за исторический период, например за последние шесть месяцев, а не одно значение.

3.2.3. Конверсии

Представьте себе, что у вас есть веб-сайт для новой религии и посетители могут подписаться на рассылку о религии за ежемесячную плату. Оформление подписки – это в некотором смысле конверсия, но нас больше интересует, когда клиенты платят. В электронной коммерции *конверсионным маркетингом* называется превращение посетителей сайта в платежеспособных клиентов.

Я знаю, что все вы полны чистыми намерениями и амбициями и собираетесь создать наилучший пользовательский опыт, а мы тут говорим о каких-то бизнес-показателях, которые вообще звучат отвратительно и вызывают желание выкинуть эту книгу! Вернемся к нашим целям, которые как раз и измеряются с помощью KPI.

Конверсия – это распространенный KPI, который определяется следующим образом:

$$\text{Конверсия} = \frac{\text{Количество достижений цели}}{\text{Количество посещений}}.$$

Данный показатель называется по-разному в зависимости от поставленной цели – это может быть коэффициент конверсии целей, коэффициент конверсии продаж и т. д.

Хотите, чтобы ваш коллега-маркетолог был так же счастлив, как Стив на рис. 3.4? Скажите ему, что вы можете вычислить коэффициент конверсии веб-сайта.

Весь онлайн-маркетинг построен вокруг понятия конверсии. Когда пользователь делает то, что вам от него требуется, то говорят о *конверсии* на сайте. Все маркетологи и создатели контента определяют это понятие по-своему.

Рис. 3.4. Довольный маркетолог Стив

Для создателей контента конверсией может являться оформление подписки, скачивание брошюры или ПО, заполнение формы обратной связи. А может быть, ваш коллега, заведя тему конверсий, начнет рассказывать о ROI (возврат инвестиций), но здесь эта тема не рассматривается.

Усложним немного: слово *конверсия* исходит из идеи *конверсионной воронки*. Она иллюстрирует путь, который проходит пользователь перед конверсией. Давайте представим, как воронка, показанная на рис. 3.5, может искать компании, такие как Amazon.

Пользователи заходят на главную страницу Amazon, что-то смотрят, и, возможно, где-то щелкают. Они могут поподробнее изучить какой-нибудь раздел, например одежду, и в конце концов купить потрясную гавайскую рубашку. Каждое событие, которое спускает пользователя на шаг вниз в воронке, называется *значимым событием*. У Netflix это, например, если пользователь смотрел фильм. На Match.com подмигивание от другого пользователя или продление подписки тоже является значимым событием на вашем пути к вечной любви.

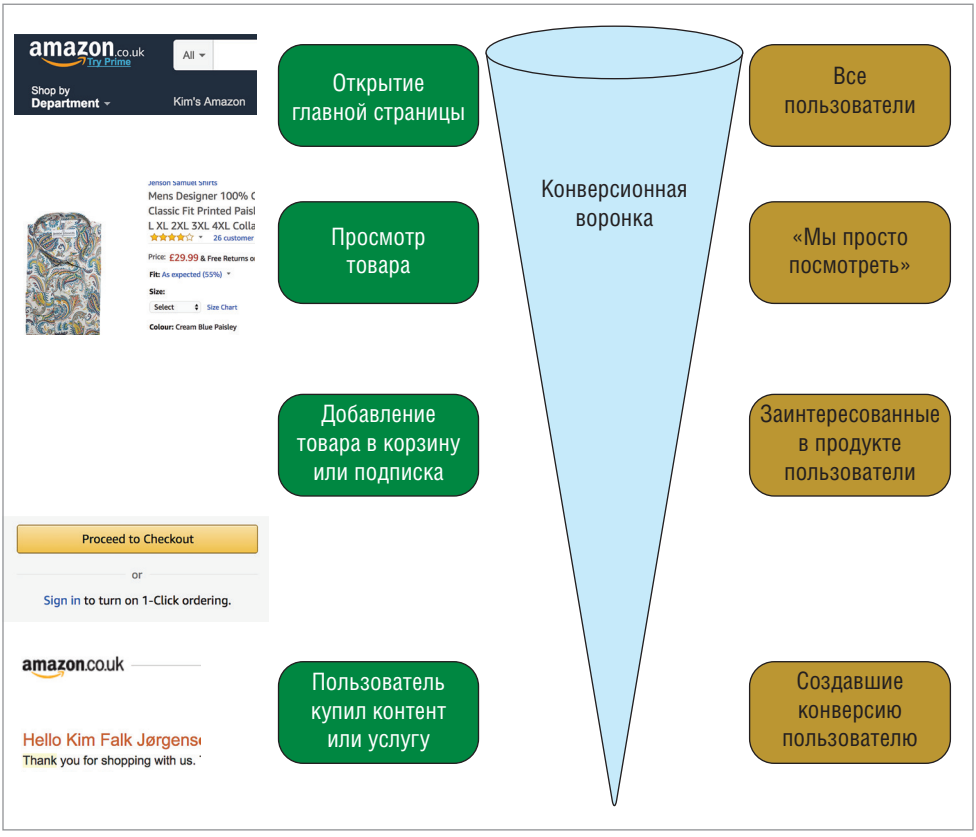


Рис. 3.5. Конверсионная воронка на примере Amazon

«Воронкой» она называется потому, что сначала она широкая, а потом сужается. То же самое происходит с посетителями сайта: на сайт заходит большое количество посетителей (широкая часть воронки), некоторые из них что-то добавят в пожелания или хотя бы посмотрят, и часть из них что-то купит (узкая часть воронки).

Слово «конверсия» пришло из жаргона маркетологов, но что, если мы посмотрим на это с точки зрения пользователя? Конверсия – это то, чего хочет пользователь. Например, пользователи заходят на Amazon с целью что-то купить или подписаться на интересную рассылку…

Amazon – это площадка продажи контента, так что, если вы идете туда что-то купить, вы ожидаете, что вам помогут добраться до лучших вещей. Позже мы рассмотрим темы, которые не совпадают с целями этих двух групп, но сейчас давайте представим сферическую ситуацию в вакууме.

На сайте MovieGEEKs сборщик данных способен регистрировать событие конверсии. А как насчет вашего сайта? Подумайте:

- Что на вашем сайте является конверсией?
- Каков коэффициент конверсии в вашей системе?

Цели могут меняться. На сайте автодилера клиент вряд ли создает конверсию при каждом посещении, но на Amazon, вероятно, большинство посещений должны превращаться в конверсию (по крайней мере, у меня так). Подумайте о том, какое изображение на рис. 3.6 подходит вашему сайту.

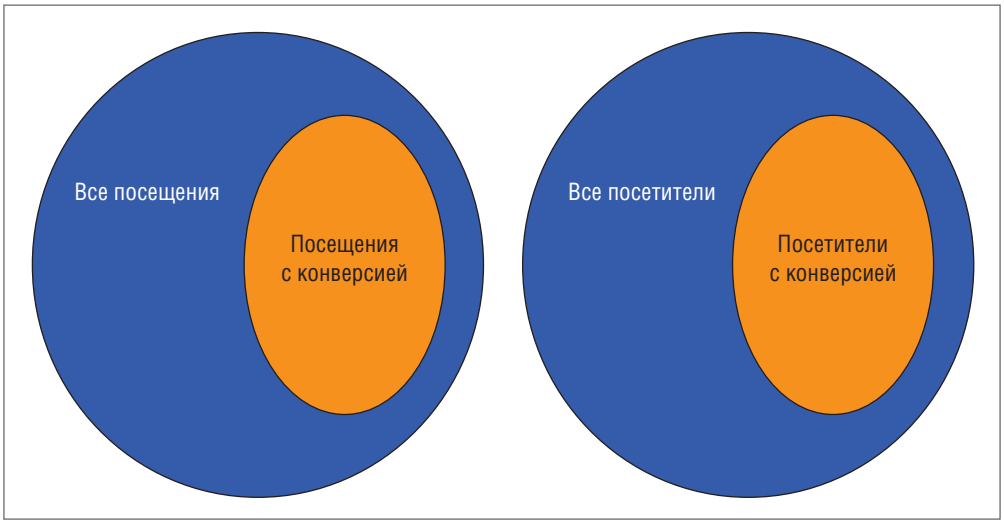


Рис. 3.6. Два взгляда на конверсию

Если вы выбрали изображение слева, это означает, что у вас сайт типа Amazon. Вы надеетесь на то, что все или почти все клиенты создают конверсию. Если вы выбрали картинку справа, то это ближе к сайту автосалона. Вы будете измерять конверсию по времени, которое пользователь проводит на сайте. Для сайта автосалона не имеет значения посещение сайта – важна только

покупка авто. В этом втором случае коэффициент конверсии вычисляется по следующей формуле:

$$\text{Конверсия} = \frac{\text{Количество достижений цели}}{\text{Количество посетителей}}.$$

MovieGEEKs – это сайт первого типа. Если посмотреть на множество посещений, то коэффициент конверсии будет вычисляться как количество сессий, в которых происходит покупка, деленное на общее число сессий. Код SQL получается следующим.

ЛИСТИНГ 3.1. Код SQL для вычисления коэффициента конверсии

```
select count(distinct(session_id)) as visits,
       count(case when conversion > 0
                  then 1 end) as conversions
from
( select session_id,
         sum(case when event = 'buy'
                  then 1
                  else 0 end) as conversion
  from collector_log
  group by session_id
) c
```

Подсчет случ ев, где сессия производит одну или больше конверсий

З прос, счит ющий сессии с конверсией

По последним сгенерированным данным, этот код возвращает 999; 97, что означает, что коэффициент конверсии составляет около 10 % (более точно 0,097). Как упоминалось ранее, оценить это значение может быть трудно, но его надо знать, так как цель рекомендаций – увеличить его.

3.2.4. О пути к конверсиям

Вы становитель счастливы, когда происходит конверсия, и цель рекомендатора состоит в том, чтобы пользователь породил конверсию. Теперь, если вы делаете запрос с целью определить типы всех событий, хранящихся в базе данных, а также подсчитать, сколько раз произошли эти события, результаты могли бы выглядеть следующим образом:

genreView	1527
details	4953
moredetails	2034
addToList	+976
buy	510

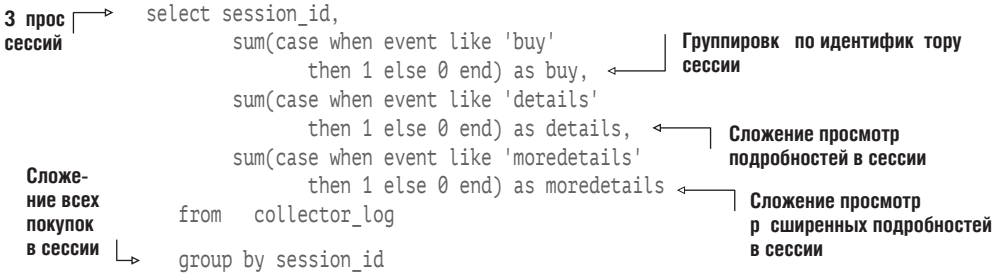
Запрос показан в листинге.

ЛИСТИНГ 3.2. Код SQL для расчета распределения событий

```
select event, count(1)
from collector_log
group by event
```

Этот запрос позволяет вам узнать, как часто происходят события каждого типа. Теперь нужно еще раз посмотреть, как часто эти события происходят в сессиях с покупками и без них. Это можно проверить с помощью листинга 3.3, в котором сессии группируются так, что для каждой сессии строка указывает число событий покупки и просмотра подробностей, которые случились в данной сессии.

ЛИСТИНГ 3.3. SQL-код, который вычисляет, сколько раз происходит каждое событие в каждой сессии



Теперь, когда вы разбили сессии в событие, можете сделать следующий шаг и посмотреть, что происходит в сессиях, где осуществляется покупка. Как вариант, можно повторно использовать предыдущий запрос и добавить фильтр, который будет выводить только сессии, когда что-то было куплено. На рис. 3.7 показан результат.

Но результата это не дает, потому что вы хотите знать, какие взаимодействия были в каждом из случаев. Важно узнать, с чем взаимодействовал пользователь, прежде чем покупать что-то.

session_id	buy	details	moredetails
441115	0	3	5
794948	0	3	1
403960	0	2	1
42483	1	3	6
794776	1	2	1
794784	0	0	0
885466	0	9	2
933590	0	12	8
794855	0	1	0
404058	0	7	2
885591	1	15	3

Рис. 3.7. Сессии, в которых происходит событие покупки

3.2.5. Путь конверсии

Путь конверсии – это путь, по которому пользователь и конкретный контент идут рука об руку к событию покупки (а потом живут долго и счастливо). Путь представляет собой последовательность страниц и действий, которые делает пользователь с момента прибытия на лендинг-страницу (первая страница, которую он увидел) до конверсии.

Это не то же самое, что воронка конверсии. Воронка конверсии состоит из предварительно определенных целей, которые должны быть достигнуты, чтобы пользователь создал конверсию. Путь преобразования – это не просто последовательность событий, а еще и последовательность страниц. MovieGEEKs не лучший иллюстративный пример, так как на нем есть лишь ограниченный набор событий. Часто у веб-сайтов предусмотрен гораздо более длинный список событий:

- просмотр содержимого;
- просмотр подробностей;
- взгляд внутрь;
- лайки;
- репосты;
- подписки на рассылки;
- щелчки по результатам поиска;
- ссылки на кампании;
- добавление в корзину;

- добавление в список избранных (об этом отдельно в следующей главе);
- оценка;
- написание отзыва;
- покупка.

Путь конверсии может быть гораздо интереснее, чем дает MovieGEEKs, на котором всего-то пять событий. Сам путь конверсии интересен, поскольку события часто являются ключевыми индикаторами того, где именно возникает связь между пользователем и контентом (эта аналогия отлично подходит для сайтов знакомств). Этим индикатором может быть что угодно – например, большинство пользователей смотрят в поиске концовку фильма перед его покупкой. Это можно как-то интерпретировать. Но как рассчитать сам путь?

Для каждого пользователя нам важно знать все сессии, в которых произошла покупка. Затем для каждой покупки нужно выполнить подсчет событий, которые произошли с купленным элементом. Осознать все это у себя в голове сложно, и от этого голова кругом. Давайте посмотрим, что мы сможем сделать.

Во-первых, нужно определить все события покупки для каждого пользователя, сессии и контента, как показано в следующем листинге. Давайте предположим, что все, что было сделано с конкретным элементом, было сделано до покупки. Это небольшая хитрость, которая бережет нас от построения сложных вычислений.

ЛИСТИНГ 3.4. SQL-код для поиска пар пользователей и контента

```
select session_id, user_id, content_id
from collector_log
where event = 'buy'
```

Этот запрос дает вам список, который вы можете присоединить к исходной таблице, получив таким образом только события, происходящие в сессии, где пользователь как-то взаимодействовал с контентом и что-то купил. Получаем запрос, показанный в листинге 3.5.

ЛИСТИНГ 3.5. SQL-код для поиска событий, которые привели к покупке

```
select log.*
from (
    select session_id, content_id
    from collector_log
    where event = 'buy') conversions
JOIN collector_log log
ON conversions.session_id = log.session_id
and conversions.content_id = log.content_id
```

Результаты показаны на рис. 3.8.

id	created	user_id	content_id	event	session_id
40094	2017-07-03 17:10:45+02	3	2096673	buy	794776
40372	2017-07-03 17:10:45+02	1	1489889	addToList	885444
40367	2017-07-03 17:10:45+02	1	1489889	genreView	885444
40360	2017-07-03 17:10:45+02	1	1489889	buy	885444
40376	2017-07-03 17:10:45+02	6	1489889	buy	42456
40323	2017-07-03 17:10:45+02	6	1489889	moredetails	42456
40615	2017-07-03 17:10:46+02	1	1291150	buy	885445
40543	2017-07-03 17:10:46+02	1	1291150	moredetails	885445
40710	2017-07-03 17:10:46+02	6	1985949	buy	42462
40806	2017-07-03 17:10:46+02	6	1489889	buy	42463
40751	2017-07-03 17:10:46+02	6	1489889	details	42463
40724	2017-07-03 17:10:46+02	6	1489889	details	42463
40715	2017-07-03 17:10:46+02	6	1489889	details	42463
41613	2017-07-03 17:10:49+02	2	2120120	genreView	403980
41234	2017-07-03 17:10:48+02	2	2120120	genreView	403980
41228	2017-07-03 17:10:48+02	2	2120120	buy	403980
41276	2017-07-03 17:10:48+02	1	3110958	buy	885463
41386	2017-07-03 17:10:48+02	1	1083452	buy	885466

Рис. 3.8. Фрагмент результата выполнения кода из листинга 3.5, в котором перечислены все события, происходящие в сессии, где была совершена покупка

Но события покупок нам не интересны, о них мы знаем и так. Окончательный запрос выглядит следующим образом.

ЛИСТИНГ 3.6. SQL-код для поиска событий, которые привели к покупке

```
select log.session_id, log.user_id, log.content_id
from (
    select session_id, content_id
    from collector_log
    where event = 'buy') conversions
JOIN collector_log log
ON conversions.session_id = log.session_id
and conversions.content_id = log.content_id
where log.event not like 'buy'
order by user_id, content_id, event
```

События покупок следует игнориров ть

В данный момент хронология событий нас не интересует, так что запрос дает все, что нужно.

3.3. Архетипы

Архетипы – это краеугольный камень ориентированного на пользователей дизайна и маркетинга. Это вымышленные люди, созданные для представления различных стереотипов, которые соответствуют группам или сегментам в пользовательском сообществе. В этом разделе представлены несколько архетипов; они не являются продуктом качественного веб-анализа, но зато охватывают контент MovieGEEKs (фотографии использованы с согласия этих людей).

На протяжении всей книги эти архетипы используются именно так, как их могли бы использовать маркетологи. Позже мы посмотрим на результаты выполнения алгоритмов и убедимся, что результаты вполне соответствуют типам. Итак, наши новые друзья:

Сара Комедия, экшн, драмы  Я не против романтических комедий, если конечно, по телевизору не идет «След» Идентификатор пользователя 400001	Джаспер Комедия, драма, экшн  Люблю посмеяться и обычно предпочитаю комедии, но посмотрел бы и драму, а порой и боевичок Идентификатор пользователя 400002
Тереза Комедия  Все, что меня развеселит, – годится Идентификатор пользователя 400003	Хелла Экшн  Тащусь от супергероев и взрывов Идентификатор пользователя 400004
Петро Драма  Чем более завернутый сюжет, тем круче Идентификатор пользователя 400005	Екатерина Драма, экшн, комедия  Драмы на первом месте, сойдет боевик, а иногда и комедия Идентификатор пользователя 400006

Каждый из этих персонажей имеет свой уникальный вкус. Для количественного определения можно подсчитать количество часов из ста, которое каждый из них потратит на тот или иной жанр. Тогда вы сможете описать каждого пользователя с помощью кортежа, содержащего число для каждого жанра. Например, вкусы Сары можно описать как 60 ч комедий, 20 ч экшна, и 20 ч драмы, т. е. ее вкус равен (60, 20, 20). Теперь сделаем это для каждого пользователя и получим цифры, указанные в табл. 3.1.

Таблица 3.1. Архетипы

	Экшн	Драма	Комедия
Сара	20	20	60
Джаспер	20	30	50
Тереза	10	0	90
Хелла	90	0	10
Петро	10	50	40
Екатерина	30	60	10

Еще, чтобы проиллюстрировать вкусы, можно построить их на схеме, как показано на рис. 3.9. Преимущество такого способа в том, что он позволяет легко найти похожие вкусы (или увидеть, что вы создали двух пользователей с такими же вкусами, как это было, когда я сделал первый график).

Некоторые компании даже делают плакаты с описаниями архетипов и требуют, чтобы все признаки описывались на основе одного из них. Это создает много странных ситуаций, потому что все сводится к обсуждению того, что будет делать один из них, как если бы это был ваш лучший друг (я даже слышал, как их пытались сватать друг к другу).

Вооружившись этими архетипами и их вкусами, перейдем к автоматической генерации данных. Это кажется слегка нечестным, но в данном случае это хороший способ, потому что вы будете «знать свои данные». Но помните, что в реальном мире все будет работать не так.

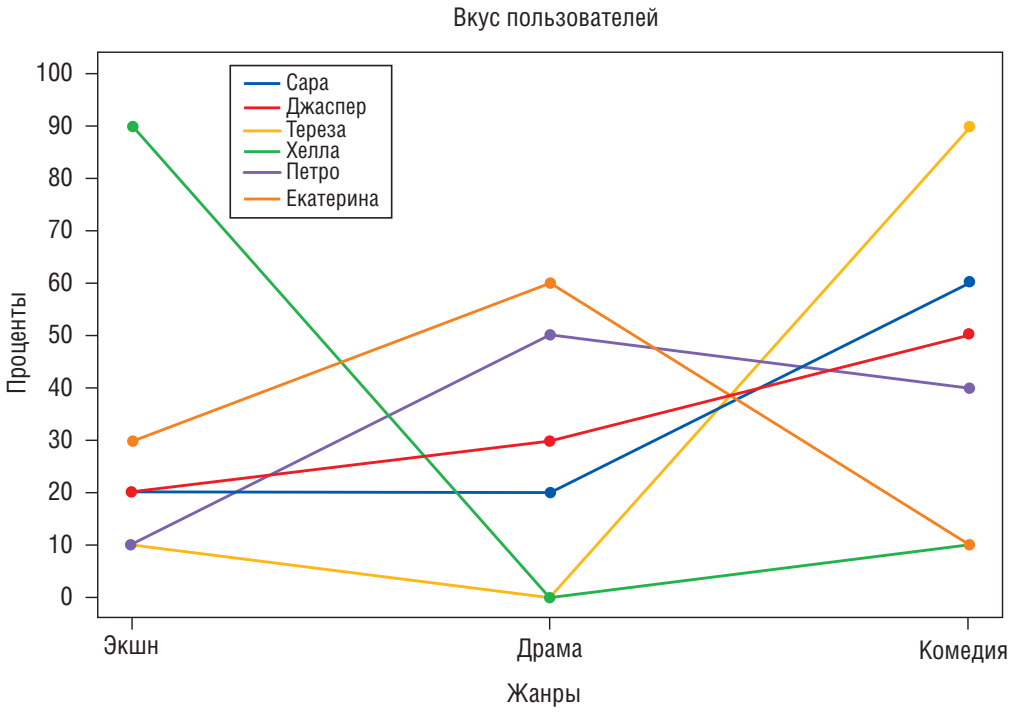


Рис. 3.9. Диаграмма вкусов пользователей

В аналитике сайта MovieGEEKs есть страница для каждого пользователя (рис. 3.10). На рисунке показано, что наша система знает о Хелле. Слева находится список наиболее высокооцененных фильмов. На диаграмме показана нормализованное представление о том, сколько фильмах Хелла оценила, и разница между средней оценкой жанра и оценками Хеллы в среднем во всех жанрах.

На рис. 3.10 видно, что Хелла оценила несколько фильмов в жанре экшн, но средняя оценка по этому жанру такая же, как и средняя по всем фильмам. Кроме того, кажется, ей нравится фантастика, хотя оценок мало.

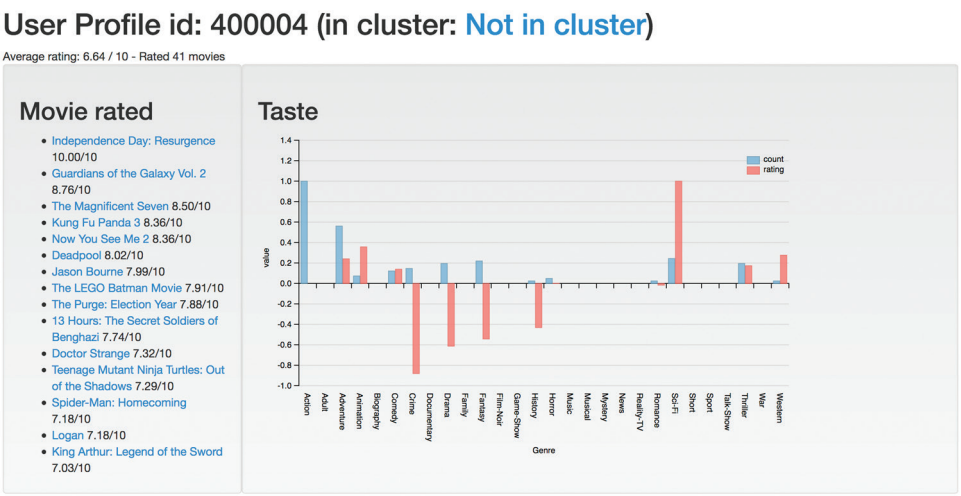


Рис. 3.10. Это профиль Хеллы. Сплошь экшн. Есть симпатия и к приключениям, поскольку многие фильмы относятся скорее к этому жанру

На графиках рис. 3.10 изображено два типа данных. Синяя (темная) полоса показывает, сколько фильмов каждого жанра Хелла оценила, а оранжевая (более светлая) – разницу со средней оценкой. Мы отложим эти данные, пока вы не прочитаете главу 4 и не запустите сценарии, упомянутые в ней. Чтобы увидеть рис. 3.10 в цвете, пожалуйста, обратитесь к электронным версиям этой книги.

3.4. Панель сайта MovieGEEKs

Напоминаем, что код сайта MovieGEEKs можно скачать из GitHub по ссылке **mng.bz/04k5**, клонировав или загрузив его. Прочитайте файл *readme.md*, в котором приведены инструкции по установке. База данных пустая, поэтому вам нужно будет запустить сценарий *late_logs.py*, краткое описание которого я дам здесь.

3.4.1. Автоматическая генерация данных в журнале

В этой и нескольких следующих главах вы узнаете о неявной обратной связи. Эту вещь нельзя потрогать или скачать, но хочется иметь на своем сайте. Поскольку на сайте MovieGEEKs мало пользователей, я создал сценарий, который заполняет часть данных и сохраняет их в базу данных, чтобы они выглядели как настоящие. Для его запуска выполните следующую команду в командной строке:

```
> Python3 populate_logs.py
```

Сценарий автоматически создает журналы для шести архетипов пользователей. Основу сценария составляет цикл, который работает в диапазоне от

нуля до числа событий, которое вам нужно. В каждой итерации выбирается случайный пользователь и фильм в зависимости от вкуса пользователя, а затем выбирается, какое он выполнил действие. Когда все будет выбрано, событие сохраняется, как показано в листинге 3.7.

ЛИСТИНГ 3.7. Выбор случайного пользователя и фильм: populate_logs.py

```
for x in range(0, number_of_events):
    randomuser_id = random.randint(0, len(users) - 1)
    user = users[randomuser_id]
    selected_film = select_film(user)
    action = select_action(user)
    if action == 'buy':
        user.events[user.sessionId].append(selected_film)
    l = Log(user_id=str(user.userId),
            content_id=selected_film,
            event=action,
            session_id=str(user.get_session_id()),
            created=datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            visit_count=0)
    l.save()
```

Выбор случ йного пользов теля

Выбор случ йного фильм по вкусу пользов теля

Выбор действия

Фильм может быть куплен лишь р з, и идентифик тор фильм доб вляется к списку пользов теля

Сохр нение журн л

Созд ние журн л и его иници лиз ция

3.4.2. Характеристики и дизайн панели аналитики

Большинство владельцев сайтов мечтает иметь панель аналитики на сайте, так что реализуем что-то подобное. У вас должна быть панель, которая показывает:

- количество посетителей и какие действия они принимают;
- количество посещений, закончившихся покупкой (коэффициент конверсии);
- самые продаваемые товары.

3.4.3. Основа панели аналитики

Панель аналитики выглядит, как показано на рис. 3.11. В верхней части приборной панели показаны KPI. Через них видно количество посетителей, зашедших на сайт за выбранный период. (Поскольку сценарий обрабатывает только шесть пользователей, он будет показывать шесть посетителей, пока не будут созданы новые пользователи. На рис. 3.11 показано семь пользователей, так как сайт создал нового пользователя для меня, когда я просматривал его.) Второй компонент показывает коэффициент конверсии, который рассчитывается от того, как много сессий закончилось покупкой, затем количество продаж в этом месяце и общее количество сессий в этом месяце.

Диаграмма слева на рисунке показывает, сколько раз происходили различные события. Диаграмма справа отображает распределение оценок.

Рис. 3.11. Панель аналитики MovieGEEKs

Каждый столбик здесь показывает, сколько фильмов получили данную оценку (это значение не появится, пока вы не запустите сценарий *populate_ratings.py*). В нижней части находится список топ-10 самых купленных фильмов.

3.4.4. Архитектура

Приложение аналитики, показанное на рис. 3.12, – это еще одно приложение в вашем проекте на Django. Как и все веб-сайты, оно состоит из двух частей: фронтенд и бэкенд. Работают обе: бэкенд запрашивает базу данных, а фронтенд визуализирует результаты. Приложение аналитики не должно быть доступно для конечного пользователя, но в этой книге мы не затрагиваем вопросы безопасности.

Как вы можете видеть, все это хранится отдельно от сайта MovieGEEKs, что позволяет легко использовать эту же архитектуру с любым сайтом[1]. Это считается лучшей практикой и позволяет использовать данное приложение для сайтов, не реализованных в Django.

В приложении аналитики несколько разных окон (или веб-страниц). Остальные страницы используются в качестве веб-API для извлечения данных. Первая, главная страница, возвращает страницу HTML, показанную на рис. 3.13.

[1] Панель аналитики отрисовывается внутри сайта MovieGEEKs, но она с ним не связана и может быть легко запущена где-то в другом месте.

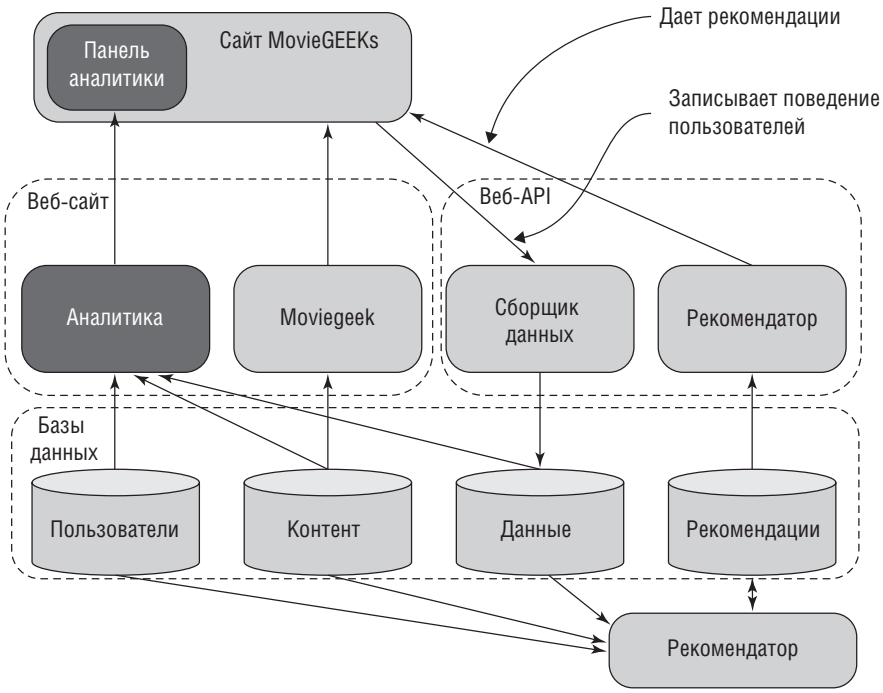


Рис. 3.12. Архитектура MovieGEEKs с выделенными аналитическими частями системы

Окно *json* возвращает данные и дает сайту данные для каждого из его компонентов.

В верхнем ряду, показанном на рис. 3.13, показано четыре ключевых показателя. За этими значениями было бы полезно следить, как уже упоминалось ранее. Первое – это количество пользователей, которые посетили ваш сайт. Следующее – известный нам коэффициент конверсии. Затем количество проданной продукции и, наконец, количество уникальных посещений (уникальное посещение означает сессии различных пользователей).

Чуть-чуть о ежемесячных данных

KPI рассчитываются за последний месяц, но могут также рассчитываться ежедневно, еженедельно или даже ежечасно, если у вас много трафика. Вы можете также использовать скользящее окно (рис. 3.14), чтобы всегда делать вычисления за последний месяц или выбрать месяц, неделю и т. д. По сути, это не так уж важно. Главное, чтобы это помогало вам отслеживать прогресс с течением времени.

В этой главе мы поговорили об аналитике и о том, как реализовать приборную панель, которая показывает простую информацию, как работает вебсайт. Такая панель будет незаменима при реализации рекомендательных систем. Теперь вы, наконец, готовы изучить, как рассчитываются оценки. После этого мы начнем создавать неперсонализированные рекомендации.

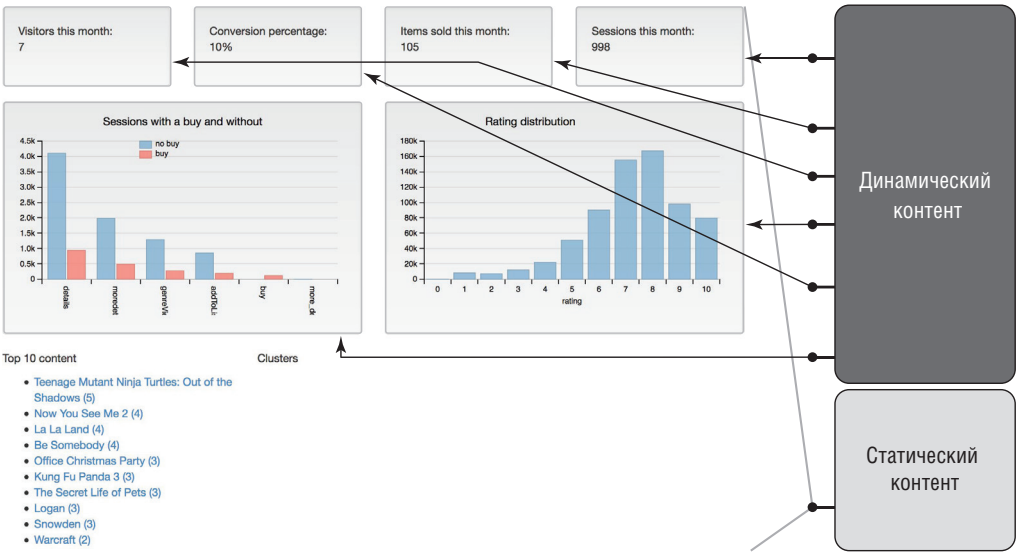


Рис. 3.13. Статический HTML-код извлекается из статического контента. Каждый компонент на приборной панели извлекает данные из своей конечной точки

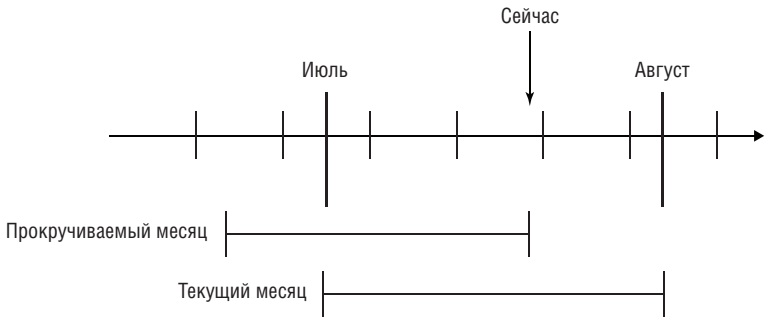


Рис. 3.14. Различные периоды времени

Резюме

- KPI – это хорошая вещь, потому что их легко оценивать и легко использовать, чтобы увидеть, есть ли у вашего сайта положительные сдвиги.
- Посетитель создает конверсию, когда он выполняет вашу цель или делает что-то, на что вы надеетесь.
- Воронка конверсии иллюстрирует последовательность шагов, которые должен предпринять ваш пользователь. Путь конверсии – это реальный путь пользователя к конверсии.
- Очень важно понимать воронку конверсии вашего сайта, чтобы вы могли осознавать, насколько пользователи близки к конверсии.
- Важно понимать и всегда иметь рабочую аналитику.

Глава 4

Оценки, и как их рассчитывать

Вас приветствует мастер создания рекомендаторов. Нажмите «далее», чтобы изучить что-то новое:

- создание матриц пользователь–элемент;
- пересмотр явных оценок с целью узнать, почему они не всегда хороши;
- погружение в тайну неявных оценок и их создания;
- кое-что о функции неявных оценок, которая превращает данные в оценки.

В этой главе мы преобразуем поведение ваших пользователей в формат, который можно будет использовать в качестве входных данных для рекомендательных алгоритмов. В начале мы рассмотрим матрицу пользователь–элемент, которую можно использовать в качестве входных данных для большинства рекомендательных алгоритмов. Затем посмотрим на явные оценки, которые пользователи добавляют сами. Неявные оценки составляют основу вашей системы, так что поговорим и о них, а именно о том, что они из себя представляют и где их взять.

На рис. 4.1 показан поток данных, о котором мы говорили ранее. Сбор данных происходит тогда, когда посетители взаимодействуют с сайтом. В этой главе мы займемся предварительной обработкой. Построение модели и рекомендаций рассматриваются в последующих главах.

Мы будем превращать поведение пользователя в оценки контента, которые вы станете использовать для создания рекомендаций в последующих главах. Этот тип оценок называется *неявным*, потому что они выводятся не напрямую.

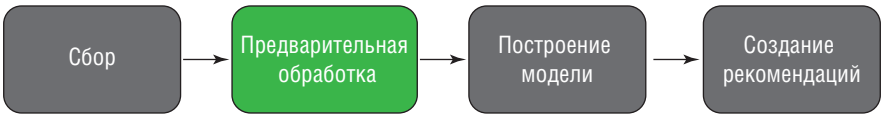


Рис. 4.1. Модель обработки данных для рекомендательных систем

Неявные оценки используются все больше и больше, потому что люди не всегда точно знают, что им нравится, и имеют тенденцию что-то смотреть, а потом рассказывать друзьям, как это было ужасно. Я смотрел фильмы на Netflix, которые потом характеризовал как ужасные. Плевался, но смотрел. Еще одна причина для использования неявных оценок заключается в том, что собирать их гораздо легче, чем явные оценки. Вот почему я большой поклонник неявных оценок, о чем вы наверняка уже догадались.

Прочитав предыдущую главу, вы должны были определить следующее:

- Какова цель вашего сайта (что должен сделать ваш пользователь)?
- Какие события ведут к этим целям?
- Сколько раз каждое из этих событий произошло?

Держите эти вещи в уме, читая эту главу. Мы начнем с того, что нужно большинству рекомендательных алгоритмов на вход, – матрицы пользователь–элемент. Идея этой главы заключается в том, чтобы собрать данные о поведении и превратить их в такую матрицу.

4.1. Предпочтения элементов пользователями

Матрицу пользователь–элемент можно рассматривать как таблицу, где в строках перечислены пользователи, а в столбцах – элементы (или наоборот). В литературе это называется *матрицей*, и мы будем придерживаться этого термина. Значение в ячейке на пересечении пользователя и элемента показывает отношение пользователя к элементу контента.

4.1.1. Определение оценок

Оценка – это, например, число звезд на Amazon.com или Glassdoor (**glassdoor.com**, сайт, где люди могут оценивать свои рабочие места) или список сердец в обзорах фильмов в местной газете. Обычно оценки – это числа от 0 до 5, которые часто превращаются в графическое обозначение и выводятся в таком виде пользователю.

Более формально оценка – это клей между пользователем, контентом и отношением пользователя к элементу контента, как показано на рис. 4.2. На рисунке изображено, что сохраняется в данных сайта после того, как Джимми посмотрел и оценил 1-й сезон «Игры престолов» четырьмя звездами. В базе данных оценки реализованы в виде таблицы, которая связывает пользователей с элементами контента.

Рис. 4.2. Связь между пользователем и контентом

4.1.2. Матрица пользователь–элемент

Пример матрицы пользователь–элемент показан в табл. 4.1.

Таблица 4.1. Матрица пользователь–элемент

	«Индиана Джонс»	«Микрокосмос»	«Мстители»	«Пит и его дракон»
Сара	4	5		
Джаспер	4		5	
Тереза	5			3
Хелла	4			5
Петро		3	4	3
Екатерина	3		3	3

Пустая клетка означает, что записей о взаимодействиях между пользователем и элементом нет. Помните, что существует разница между пустой ячейкой и ячейкой, в которой записан ноль. Ноль – это оценка, а пустая ячейка означает отсутствие оценки.

Пустых клеток может быть и немного, но именно в них заключается суть большинства традиционных рекомендательных систем. Большинство рекомендательных систем пытается предсказать, что пользователь поставил бы тому или иному элементу. Если пустых ячеек слишком мало – пользователь исчерпал весь контент, а если слишком много, то у рекомендатора не будет достаточно данных, чтобы понять, что нравится пользователю.

Если у вас с кем-то зашла беседа о матрице пользователь–элемент (обычное дело, не так ли?), то стоит поговорить о проблеме разреженности. Это немного похоже на разговоры о кормлении детей при общении с новоиспеченными родителями, которые могут говорить об этом часами.

Проблема разреженности

Матрица пользователь–элемент не всегда заполнена так густо, как показано в табл. 4.1. На самом деле непустые клетки встречаются редко, потому что у многих интернет-магазинов очень много и пользователей, и элементов. При этом большинство пользователей покупают один-два товара, и матрица тогда будет выглядеть примерно как на рис. 4.3.

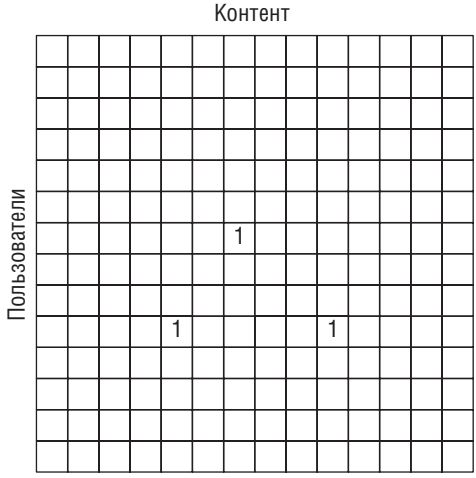


Рис. 4.3. Разреженная матрица пользователь–элемент

Как мы увидим позже, матрица пользователь–элемент является входными данными для рекомендательных алгоритмов. Конечно, не хочется иметь матрицу, которая выглядит как датский пляж в середине зимы (и я знаю, о чем говорю, см. рис. 4.4).

В главе 8 мы поговорим о семействе рекомендательных алгоритмов *совместной фильтрации*. В этом семействе алгоритмов матрица пользователь–элемент используется, для того чтобы найти схожих пользователей. Если матрица разрежена (или пуста), она дает мало информации и это затрудняет вычисление рекомендаций. В главе 5 мы узнаем, как вычислять рекомендации, даже если у вас разреженная матрица. Сейчас давайте сосредоточимся на том, как заполнить эту матрицу либо *явными оценками* (добавлены вручную пользователем), либо *неявными оценками* (рассчитываются из данных, которые вы собираете).

Рис. 4.4. Датский пляж в холодный зимний день

4.2. Явные или неявные оценки

Данные из матрицы оценок, показанной здесь ранее, используются для примеров в этой главе. Приложение извлекает оценки из двух источников, наиболее важным из которых является банк данных MovieTweetings[1]. Другая часть рассчитывается на основе поведения пользователя, данные о котором генерируются так, как описано в главе 3.

В реальной работе эти данные могут поступать из оценок, явно добавленных пользователями, если у сайта есть система оценок, а также из данных поведения пользователя (или и того и другого). На рис. 4.5 показан пример разницы между явными и неявными оценками.

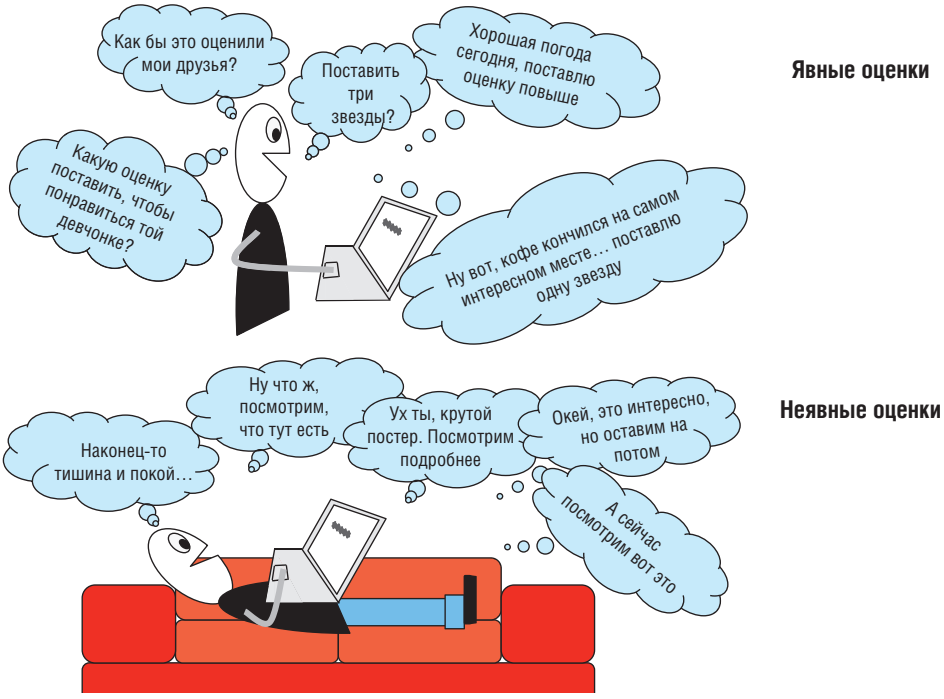


Рис. 4.5. Разница между явными и неявными оценками. Много факторов влияет на то, что вы оцените и что вы в конечном итоге будете смотреть

На сайтах с фильмами типа MovieGEEKs, если пользователь что-то купил, а затем поставил оценку, такая оценка, вероятно, заслуживает доверия. Или нет? А если мне понравились фильмы, которым я поставил низкую оценку? Стоит ли доверять ей?

Эти вопросы зависят от задачи, так что дать универсальный ответ непросто. Но если пользователь продолжает покупать фильмы, которым ставит низкие оценки, то, вероятно, надо подсказывать ему такие же фильмы.

[1] Подробнее: **github.com/sidooms/MovieTweetings**.

Но пользователь HBO (стриминговый сервис с подпиской) оценивает то, что доступно только на HBO, не видя даже сам фильм. Стоит ли доверять ему? Всегда нужно критически смотреть на то, что показывают нам данные.

Правда о вкусах пользователей

Важно помнить, что собранные вами данные – реальны, и вы должны придерживаться объективного мнения о том, что пользователи делают на сайте. Превращение поведения в оценки субъективно и может различаться в разных задачах и алгоритмах.

4.2.1. Как мы используем доверенные источники для составления рекомендаций

Как вы думаете, ваши коллеги – надежный источник мнения? Некоторые сайты тоже задействуют пользователей для продажи вещей, так как люди стараются выглядеть лучше и перещеголять конкурентов. Например, кому-то пришла в голову сумасшедшая идея, и он написал книгу о рекомендательных системах. Он был начальником крупной компании (не все из этого правда), и тогда он сказал, что все его 2181 сотрудник должны были дать положительный отзыв о книге, или они будут уволены. Тогда страница Amazon может выглядеть, как на рис. 4.6.

Это утрированный пример, но я видел ситуацию, когда человек написал книгу, а затем подарил ее всем своим сотрудникам. А еще иногда люди ставят плохие оценки, если на почте порвался пакет. Я не говорю, что вы не должны доверять обзорам на сайте, но стоит подумать о том, что заставляет пользователей ставить хорошие или плохие оценки. Независимо от того, явные это оценки или нет, они могут быть ненастоящими.

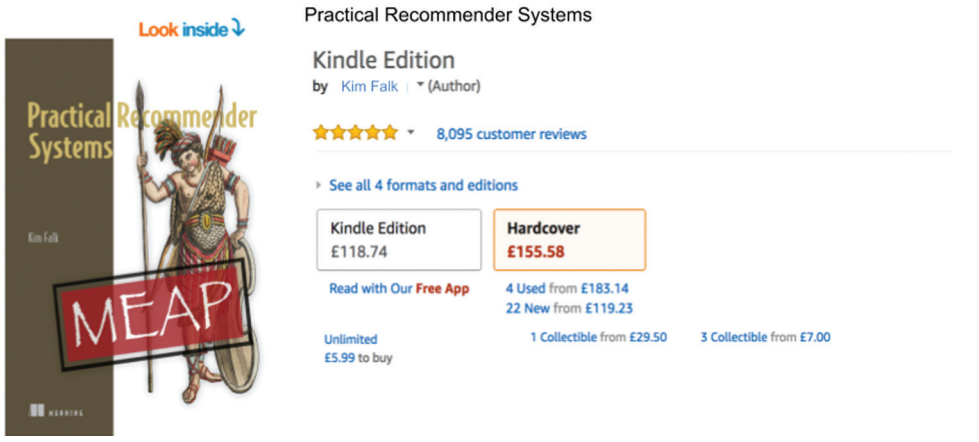


Рис. 4.6. Эта книга на Amazon (страница ненастоящая). Читатели могут поразмыслить, где и что я копипастил

4.3. Переоценка

Когда пользователь вручную дает оценку элементу контента, это называется *явной оценкой*. Самый простой способ заполнить матрицу пользователь–элемент – попросить пользователей сделать это самостоятельно. Вот только они этого не делают, даже если вы даете им возможность.

Сколько людей оставляют отзывы книгам или фильмам? И даже если они делают, вы не можете всегда быть уверены, что оценки отражают их истинное мнение. Люди подвержены влиянию того, что думают и делают их близкие. Что именно оценивает пользователь? В следующий раз, когда будете ставить оценку, подумайте вот о чем: вам не понравился весь товар целиком или какая-то его часть? Может быть, вам понравился фильм, но обложка на DVD была некрасивой, и вы поставили низкую оценку. Можете ли вы раскритиковать хорошую газонокосилку за некрасивое крепление винта?

Когда вы закончите это читать, я надеюсь, вы не только начнете писать рекомендательные системы, но и разрекламируете эту книгу. Для этого можно поставить ей хорошие оценки, например, на Amazon. Это будет явная оценка. Такую оценку можно непосредственно перенести на матрицу[1]. Но если пользователи ленятся или упрямятся, то с их явными оценками работать не стоит.

Сайты вроде TripAdvisor, Glassdoor и другие подобные существуют только за счет оценок пользователей. Напоминаю, что даже если неявные оценки в целом более точно отражают мнение пользователя, явные оценки все еще используются. Мы вернемся к явным оценкам позже, а пока сосредоточимся на неявных.

4.4. Что такое неявные оценки?

Неявные оценки выводятся из мониторинга поведения людей. Звучит немного страшно, не так ли? Ваша задача здесь – облегчить информационную перегрузку и помочь пользователям, а не преследовать их и не заставлять покупать что-то.

Большинство согласится с тем, что, когда пользователь покупает продукт, это означает, он ему нравится. Это можно расценивать как позитивную оценку данного продукта пользователем. Это и есть неявная оценка. То же самое справедливо, когда пользователь смотрит контент или дополнительную информацию о продукте. Это положительные примеры взаимодействия пользователей и контента. А вот возврат товара – это пример негативной неявной оценки.

Для расчета неявной оценки нужно взять все события, произошедшие между каждым пользователем и каждым элементом, и определить число, показывающее, насколько сильно пользователю нравится этот контент. То есть мы пытаемся предугадать оценку пользователя по его взаимодействию с контентом (просмотр, покупка и т. д.). Вы можете определить, что означает каждое событие, но действия пользователей часто могут быть интерпретированы

[1] Хотя обычно оценки еще нормализуются.

по-разному, поэтому желательно создать систему, которая позволяет подстраивать алгоритмы вычисления оценки.

Когда данные будут собраны, у вас будет много способов использовать их. Известный рекомендательный алгоритм элемент–элемент сайта Amazon создает рекомендации, основываясь только на данных о покупках, и вуаля – у сайта более чем 200 млн активных пользователей[1]. Некоторые используют историю просмотра всех пользователей и рекомендуют подобные страницы. Какой подход лучше подходит для вашего сайта, зависит от того, что вы продаете и кто ваши клиенты. После прочтения этой главы вы получите более четкое представление о том, какой путь выбрать.

На некоторых сайтах само понятие «покупки» вовсе отсутствует. Сайт The New York Times, например, использует историю просмотров, на основе ее рекомендуя для вас новый материал. На рис. 4.7 показаны рекомендации с первой страницы сайта (**www.nytimes.com**).

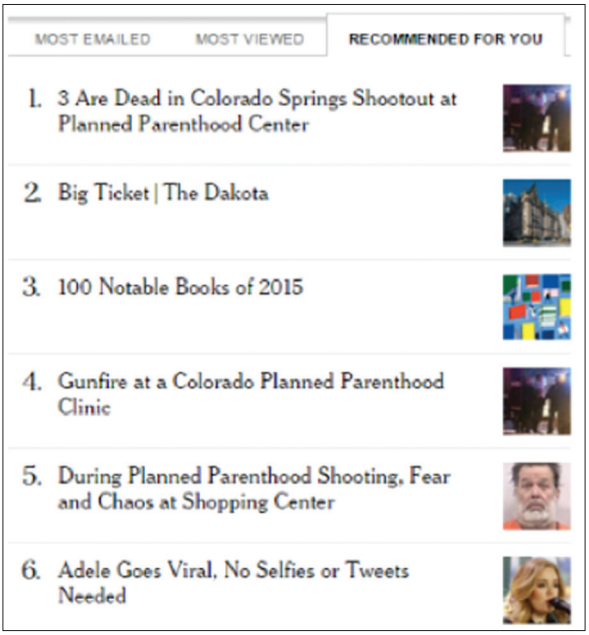


Рис. 4.7. Рекомендации с сайта The New York Times

The New York Times использует неявные оценки, так как системы явных оценок на сайте нет. Более того, даже если пользователям дать возможность ставить оценки, причина оценки будет неизвестна. Если она минимальная – что именно не понравилось пользователю? Может, тема, а может, стиль написания или сам сюжет?

[1] На Amazon, безусловно, используется нечто большее, чем просто «покупка», но в конкретном алгоритме из статьи «Рекомендации Amazon.com: совместная фильтрация элемент–элемент» (. IEEE Internet Computing, ст.7 н.1, стр 76-80, январь 2003) используется только это событие. Но 2003 год был очень давно. Вы можете просмотреть статью по ссылке **www.cs.umd.edu/~samir/498/Amazon-Recommendations.pdf**.

Важно отметить, что даже элемент с высокими оценками не обязательно будет лучшим вариантом рекомендаций. Я, например, люблю мохито, который делают в определенном кафе в Италии. Но это не значит, что я захочу мохито на завтрак, даже если вечером я несколько раз поставил высокие оценки.

Иногда стоит разделять данные на временные интервалы и делать разные рекомендатели для каждого из них. Но это компромисс между точностью рекомендаций и разреженностью данных.

Всегда стоит помнить о *релевантности*. Не во всех приложениях есть система оценок, и не всегда она вообще нужна. Еще один хороший пример (в дополнение к The New York Times) – это eBay. Какую информацию получит eBay от вашей оценки редкого ланчбокса с Чудо-женщиной 1984 года? Он может вообще быть один на весь сайт. Полезная информация здесь – ваша любовь к комиксам, коллекционированию или ланчбоксам.

У многих других сайтов есть контент, оценивать который нет смысла, но эти сайты все равно есть рекомендации. Это могут быть государственные сайты с информационными документов или сайты агентств недвижимости. Туда же отнесем и образовательные сайты, которые являются еще одной областью, где в неявных оценках заложено больше информации, чем в явных.

4.4.1. Предложения людей

Немалая часть вычислительной мощности тратится на вычисления того, как предложить людям других людей. Одно из наиболее известных мест, которые это делают, – это старый добрый LinkedIn, который также утверждает, что они первые придумали эту идею (за исключением сайтов знакомств разве что).

У LinkedIn есть функция **Возможно, вы знакомы**, где предлагаются люди, которых можно добавить к вашей сети. Это пример сайта, где было бы неуместно вводить явные оценки. Но таким сайтам, как LinkedIn (или Facebook), все равно требуются вычисления для составления рекомендаций.

Мы уже идем по краю между рекомендательными системами и областью интеллектуального анализа данных.

4.4.2. Что учитывать при расчете оценок

В этом разделе мы рассмотрим несколько моментов, важных для расчета оценок. Какой использовать подход – зависит от того, какие у вас есть данные и какой у вас тип сайта.

В главе 2 вы видели, что событие покупки возникает до оценки. Правда заключается в том, что вы ничего не знаете об этом событии, и это самая трудная задача. Чтобы справиться с задачей, нам придется предположить, что пользователь покупает товар, потому что он кажется ему хорошим. Элемент может оказаться плохим, но в целом люди покупают вещи, потому что они хотят купить их. Покупка может оказаться подарком для мачехи пользователя, но если уж пользователь купил что-то, так почему бы не рекомендовать что-то еще для следующего подарка? Основываясь на этом, посмотрим на данные событий покупки.

Двоичная матрица пользователь–элемент

Используя события покупки, вы можете составить простую матрицу пользователь–элемент, применяя эту формулу:

$$r_{ij} = \begin{cases} 1, & \text{где пользователь } i \text{ покупает товар } j \\ 0 & \text{в противном случае} \end{cases}.$$

Каждая ячейка в матрице пользователь–элемент содержит 1, если пользователь (*i*) купил товар (*j*), и 0 – в противном случае. Фрагмент такой матрицы приведен в табл. 4.2 (аналогичную матрицу можно составить так: 1, если пользователю понравился фильм, и 0 – в противном случае).

Таблица 4.2. Двоичная матрица пользователь–элемент

Пользователи	Фильмы			
	«Индиана Джонс»	«Микрокосмос»	«Мстители»	«Пит и его Дракон»
Сара	1	1	0	0
Джаспер	1	0	1	0
Тереза	1	1	0	1
Хелла	1	0	0	1
Петро	0	1	1	1
Екатерина	1	0	1	1

Больше всего принцип «купил/не купил» используется на Amazon. Я регулярно захожу туда в поисках книг по Python или анализу данных, но часто в конечном итоге перехожу на сайты Manning или O'Reilly, так как они часто дают свободный доступ к большему количеству книг или скидки на 50 %. Я купил на Amazon несколько книг, которые не могу прочитать на моем компьютере, а могу только на планшете, и это сводит меня с ума.

Поскольку я часто бываю на Amazon, Amazon видит по моей истории просмотра, что я заинтересован в Python и анализе данных, но он рекомендует только те книги, которые я уже купил, как показано на рис. 4.8. В последнее время я выбрал несколько книг по Microsoft Azure в основном потому, что они были бесплатными.

Я купил свою первую книгу на Amazon примерно в 2000 году и с тех пор приобрел еще много. Но все ли они отражают мой вкус? Раньше я был разработчиком на Java, но теперь этот этап позади. Я больше не заинтересован в книгах по Java, так что надеюсь, что элементы, купленные недавно, будут для рекомендатора важнее, чем то, что было15 лет назад. В следующем разделе мы поговорим об этом.

Временной подход

Использование двоичной матрицы делает все черным или белым. Но большинство сайтов хотят иметь более тонкую настройку, чтобы рекомендатор мог получить полнее представление о том, что нравится пользователям.

Говорят, ничто не сравнится с вашей первой любовью. Но у рекомендаторов это не так. Здесь наибольшее внимание следует уделять недавним событиями. Таким образом, чтобы сделать матрицу более гибкой, нужно использовать функцию, зависящую от даты покупки. Можно даже учесть время изготовления элемента, если хотите. Например, совершенная пять минут назад покупка элемента, который был произведен (или добавлен в каталог) пять минут назад, будет иметь более высокое значение, чем покупка старого продукта те же пять минут назад или покупка годичной давности.

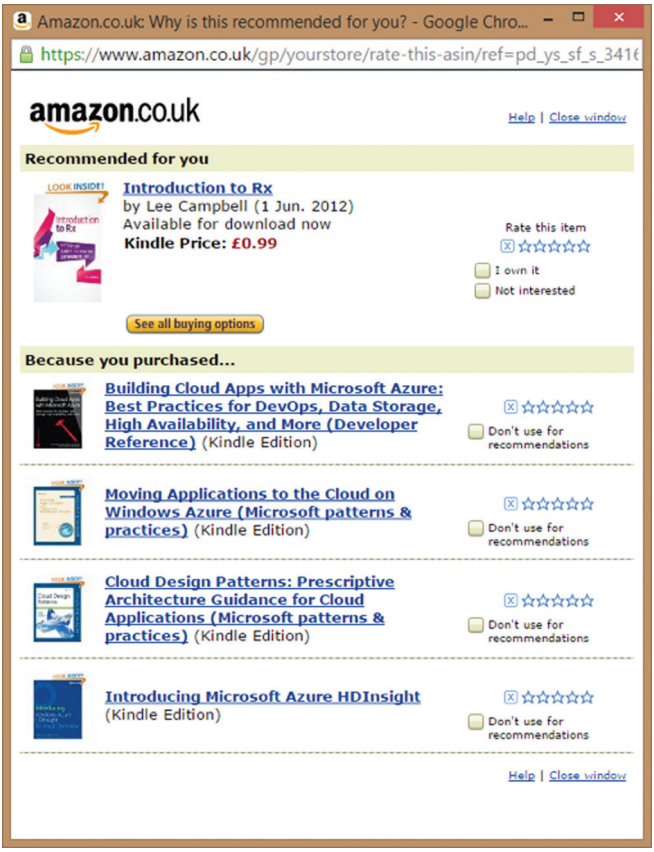


Рис. 4.8. Amazon показывает рекомендации в зависимости от того, что вы покупаете

Такой подход поощряет продвижение новых элементов. Даже если пользователь покупает много старых мелодрам и купил один новый боевик – он победит благодаря новизне.

Алгоритм Hacker News

На сайте Hacker News (**news.ycombinator.com**) используется несколько схожий алгоритм, который выдвигает вперед недавние события и меньше продвигает старые. Этот тип алгоритмов называется *алгоритмами временного затухания*.

Когда Hacker News в последний раз рассказывал, как он работает, в нем для расчета оценки статьи использовалось следующее уравнение. Формула берет рейтинг (количество «пальцев вверх» минус количество «пальцев вниз») и делит его на элемент времени распада, используя термин *гравитация*, который показывает, насколько быстро затухает рейтинг элемента:

$$\frac{\text{рейтинг} - 1}{(\text{время распада в часах} + 2)^{\text{гравитация}}}.$$

Был момент, когда Hacker News определял гравитацию равной 1,8, но он постоянно подстраивает алгоритм. Вкусы людей меняются, и то, что когда-то было лучшим, сейчас может уже и не быть таковым. Это подтверждает то, что старые события должны цениться меньше, чем новые.

Поведенческий подход

Большие компании вроде Amazon, наверное, утонули бы в данных, если бы они использовали данные об иных действиях помимо покупки. Но для большинства сайтов показанная ранее бинарная таблица была бы совершенно пустой или заполненной нулями. Вот почему хорошей идеей было бы немного расширить горизонт и добавить другие события, помимо событий покупки.

Использование нескольких видов событий – это уже посложнее, потому что вам нужно количественно оценить каждое из них. Когда значение каждого события будет определено, любая запись в матрице пользователь–элемент может быть вычислена на основе всех событий, которые произошли между пользователем и элементом. Именно с этим подходом мы будем работать, слегка приправив его временным подходом.

4.5. Расчет неявных оценок

Учитывая знания, которые вы приобрели после работы с данными в предыдущих главах, что вы можете сказать о событиях? Можете ли вы сказать, что действия, которые привели к покупке, означают, что клиент был ближе к ней? Трудно сказать, так ли это всегда, но в большинстве случаев это, вероятно, правда. Таким образом, вы будете работать с событиями, собранными на сайте MovieGEEKs, начиная с событий покупки.

В этом сайте есть только события просмотра подробностей и расширенных подробностей, которые используются для расчета неявных оценок. Удачно, что тут все довольно просто. У большинства сайтов предусмотрено намного больше типов событий. Теперь, вероятно, стоит остановиться и подумать о том, что вы рассчитываете, потому что в некотором смысле вы рассчитываете не совсем оценку в буквальном смысле. Давайте посмотрим на пример.

Моя изначальная цель изучения итальянского языка заключалась в том, чтобы почитать книги Умберто Эко (рис. 4.9) в оригинале. Это была глупая затея, потому что я так и не потрудился прочитать что-либо из его книг (даже те, которые были переведены), прежде чем отправиться на вечерние занятия по итальянскому языку. Но в тот момент это казалось хорошей идеей. Десять лет спустя я женат на итальянке и неплохо говорю по-итальянски, хотя и недостаточно хорошо, чтобы свободно читать книги Эко.

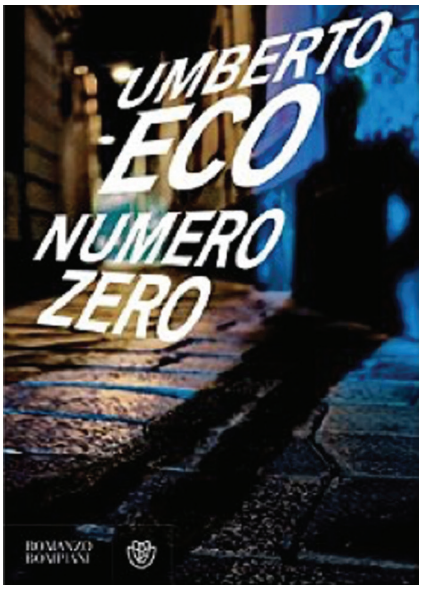


Рис. 4.9. Книга Умберто Эко, итальянского писателя и литературного критика, который умер в 2016 году

Но я все равно обязательно покупал его книги, когда выходила новая, и пробовал их почитать, даже если не был уверен в том, понравится ли мне это. Тем не менее есть и другие писатели, которых я люблю и рекомендую другим.

Если у вас сайт с книгами, вы хотите знать, какие книги я бы потенциально купил независимо от того, какую оценку я бы им поставил. То есть вы рассчитываете число, которое указывает на то, насколько вероятно, что ваш пользователь совершит покупку, а не оценку пользователя. Помимо знаний о моей одержимости книгами Эко, этот пример может также дать представление о том, что вы хотите найти.

4.5.1. Просмотр поведенческих данных

Давайте возьмем пример с сайта MovieGEEKs. Джаспер (пользователь 400002) обожает Джима Кэрри (рис. 4.10). Он думает о покупке фильма «Эйс Вентура». Он смотрит и думает, «ну не знаю, дороговато». Потом смотрит снова и снова. Наконец, он решает почитать подробности, чтобы точно убедить себя в покупке, что он и делает. Список событий приведен в табл. 4.3.

Таблица 4.3. Собранные данные Джаспера

Идентификатор пользователя	Элемент контента	Событие
2	*«Эйс Вентура: Когда зовет природа»*	Щелчок по кнопке **Подробности**
2	*«Эйс Вентура: Когда зовет природа»*	Щелчок по **Подробности**
2	*«Эйс Вентура: Когда зовет природа»*	Щелчок по **Подробности**
2	*«Эйс Вентура: Когда зовет природа»*	Щелчок по **Дополнительные подробности**
2	*«Эйс Вентура: Когда зовет природа»*	Щелчок по **Купить**

Рис. 4.10. Джаспер любит Эйса Вентуру

Вы знаете, что он купил фильм. Но если бы я остановил рассказ до покупки, вы бы думали, что если пользователь просто несколько раз просмотрел информацию о фильме, то этого уже достаточно, чтобы делать выводы. Если Петро случайно щелкнет по «Эйсу Вентуре», то это действие не должно ничего значить. Но если он вернется, то уже другое дело.

В этом примере я не рассматривал временной фактор. Если щелчки были сделаны в течение короткого периода времени, они могут не означать то же самое, что щелчки, сделанные в течение нескольких дней. Суть в том, что нужно добавлять правила для ваших неявных рекомендаций, например:

- покупка => высшая оценка;
- просмотр подробностей и расширенных подробностей => хорошая оценка;
- пара просмотров подробностей => хорошо;
- один просмотр => неясно.

На панели MovieGEEKs есть диаграмма, которая показывает, какие действия наиболее часто приводят к событию покупки. Вы также можете получить похожую картину в отношении того, как рассчитывать неявные оценки. Рисунок 4.11 является повторением части рис. 3.2. Мы пошли на небольшой трюк, так как график показывает автоматически сгенерированные данные, поэтому события распределены так специально. Но в реальной системе такой график будет хорошо иллюстрировать, какое значение лучше присвоить каждому событию.

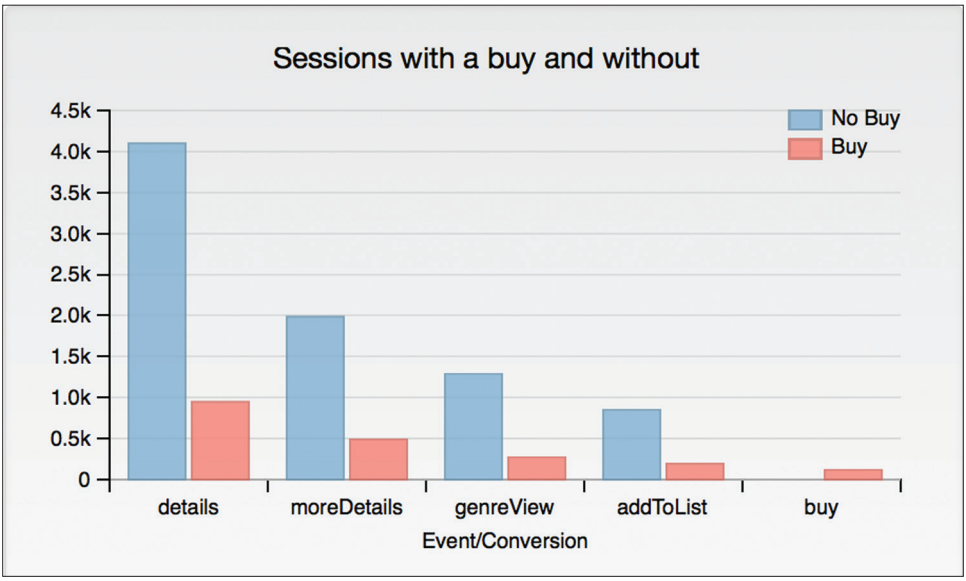


Рис. 4.11. Диаграмма, показывающая, сколько событий каждого типа произошло в сессиях с покупкой и без нее

Чтобы подвести итоги этого исследования и помня о Джаспере, вы можете определить функцию неявных оценок, которая будет выводить число, показывающее, сколько пользователей *u* будут заинтересованы в покупке элемента *i*. Если говорить точнее, то нам интересно узнать, насколько пользователь *u* близок к покупке элемента *i*, поэтому вы можете использовать эти знания, чтобы найти похожие элементы, которые пользователь может купить вместо или вместе с элементом *i*. Имея это в виду, рассмотрим следующую неявную оценку элемента *i* для пользователя *u*:

$$IR_{\text{i,u}} = \left(w_1 \times \#\text{событие}_1\right) + \left(w_2 \times \#\text{событие}_2\right) + \ldots + \left(w_n \times \#\text{событие}_n\right),$$

где:

- $IR_{i,u}$ – неявная оценка;
- *#событие* – это сколько раз произошел определенный тип события;
- $w_1 \ldots w_n$ – это весовые коэффициенты, заданные вами на основе предыдущего анализа (их можно подстраивать в процессе).

Должны ли все события иметь положительный вес?

Если действия чаще происходят в сессиях с конверсией, чем в сессиях без них, вы можете дать таким событиям положительную оценку или вес. Все взаимодействия между пользователем и контентом на вашем сайте и в целом представляют собой положительный признак, так как пользователи интересуются контентом.

Можно найти и исключения, например дизлайки. Их внедрил Netflix, перейдя от оценок к пальцам вверх или вниз[1]. Палец вниз может указывать на то, что пользователь не хочет видеть подобный контент, что может быть истолковано только как добавление контенту отрицательного балла или веса.

Расчет веса

Давайте создадим функцию путем добавления веса в соответствии с тем, что вы узнали в примере с Джаспером и Эйсом Вентурой. Как я уже говорил несколько раз, важно не думать, что однажды сделанное будет работать вечно.

> **ПРИМЕЧАНИЕ.** Хорошо работать с весами и функцией, когда у вас есть полноценный и рабочий рекомендатор, а затем посмотреть, как будет выглядеть результат.

Сначала следует определить следующее:

- $событие_1$ = покупка;
- $событие_2$ = расширенные подробности;
- $событие_3$ = подробности.

Теперь давайте попробуем вывести веса для этой функции так, чтобы покупка имела максимальную оценку. Пусть высший балл будет 100 (вы сможете нормализовать его позже). В табл. 4.4 показаны ваши предположения, которые могут быть превращены в веса.

Таблица 4.4. Веса событий

События	Интерпретация	Пример значения
Покупки	Высший балл	$100 = (w_1 \times 1)$
Один или несколько щелчков по кнопке **Подробности** и **Расширенные подробности**	Очень хороший	$80 < (w_2 \times 1) + (w_3 \times 3)$
Несколько щелчков по **Подробности**	Хороший балл	$50 < (w_3 \times 3)$
Щелчок по **Расширенные подробности**	Неопределенно	$(w_3 \times 1) < 50$

w_1 – это вес для покупки, w_2 для расширенных подробностей, а w_3 – подробности.

[1] Том Вандербилт «Теперь Netflix состоит из пальцев» **mng.bz/2GV8**.

Имея список уравнений, мы получаем задачу оптимизации, которая может быть решена математически, но уравнения исходят из предположения, так что мы не можем доверять им, как Библии. Чтобы решить эти уравнения, нужны весовые значения (w_x), подходящие под предыдущие правила:

- $w_1 = 100$;
- $w_2 = 50$;
- $w_3 = 15$.

Если вставить эти веса в предыдущую формулу, она будет выглядеть следующим образом:

$$IR_{i,u} = (100 \times \#\text{событие}_1) + (50 \times \#\text{событие}_2) + (15 \times \#\text{событие}_n).$$

В качестве теста давайте попробуем вычислить неявную оценку Джаспера для Эйса Вентуры:

$$IR_{i,u} = (100 \times 1) + (50 \times 1) + (15 \times 3)$$
$$IR_{i,u} = 195.$$

Если вы рассчитаете неявную оценку для фильма, который Джаспер нашел интересным и нажал на **Подробности**, то:

$$IR_{i,u} = (100 \times 1) + (50 \times 1) + (15 \times 3).$$

Мы будем нормировать эти оценки, а это значит, что вы будете корректировать их так, чтобы они лежали между 1 и 10. Стоит подумать, нужно ли отрезать какие-то события, если их количество в какой-то момент перестает давать больше информации. Это может случиться, если кто-то смотрит подробности больше, чем определенное количество раз. Подумайте об этом: если пользователь нажмет на **Подробности** одного элемента три раза, будет ли разница между этим и 10 нажатиями?

Для реализации отсечки может быть полезно заменить `#eventn` на `min(#eventn,relevant_maxn)`. `relevant_maxn` будет различным для каждого типа события. Формула возвращает число раз, которое происходит событие, если его значение не превышает $\texttt{relevant\_max}_n$.

Использование большего количества релевантных данных

Первый раз, когда пользователь возвращается на ваш сайт, ваши оценки будут основаны главным образом на данных предыдущих просмотров, а не покупок, поэтому рекомендации будут основываться на меньшем количестве вещей. Когда пользователь взаимодействует с сайтом и делает покупки, контент, с которым взаимодействует пользователь, уточняется до конкретных вкусов пользователя.

Этот можно было бы считать проблемой машинного обучения.

Сегодня многие компании тратят много энергии, пытаясь предсказать, какие пользователи готовы покупать и когда. У вас есть аналогичная проблема: нужно предсказать, насколько вероятно, что клиент собирается купить определенный продукт, который просматривал потенциальный покупатель.

Нельзя сказать наверняка, существует ли связь между взаимодействием пользователя с сайтом и его покупками, но это кажется правдоподобным. Если да, то определенные цифры или вектор могут быть умножены на собранные данные, чтобы получить *вероятность*, что указывает на то, насколько близко клиент находится к покупке чего-то. Это довольно близкое приближение того, что вы пытаетесь вычислить. Если вы определили это в терминах машинного обучения, что существует функция:

$$Y = f(X) + \varepsilon,$$

где Y представляет собой истинное предсказание близости пользователя к покупке конкретного элемента. Теоретически это может быть вычислено с помощью вставки ваших данных в журнал признаков. Также мы включим параметр *шума* – ε, – указание на то, что вы не можете получить всю информацию из признаков, так что у вас будет чувство неопределенности независимо от того, насколько близко ваша функция f касается расчета реальной оценки. Целью многих алгоритмов машинного обучения является использование данных для аппроксимации функции f, и я призываю вас попробовать ее с вашими данными.

Какой вид машинного обучения можно применить к этой проблеме? Вы хотите предсказать неявную оценку на основе различных событий. Чтобы сделать это, вы можете только попытаться предсказать, приведет серия событий к покупке или нет. Чтобы сделать это, вам понадобится классификатор.

Для начала подойдет наивный байесовский классификатор. Он не только решает задачу классификации, но и дает вероятность того, насколько он уверен в классификации. В этом случае вы можете использовать вероятность того, что классификация предскажет покупку. Для получения более подробной информации о том, как использовать наивный байесовский классификатор, смотрите главу 5 книги Джеффа Смита «Реактивные системы машинного обучения» (Manning, 2017).

Если вас интересует только машинное обучение, то читайте дальше. Некоторые из алгоритмов, используемых для расчета рекомендаций, представляют собой машинное обучение. О них мы подробно поговорим в этой книге. Далее давайте посмотрим, как можно реализовать вычисления неявных оценок.

4.6. Как реализовать неявные оценки

Ну хватит топтаться на месте – нам нужен код! Давайте начнем с обзора того вида, где вы хотели бы реализовать эту функцию на MovieGEEKs, а затем перейдем к объяснению того, как это сделать. Рассмотрим следующее:

- получение данных;
- расчет оценки;
- просмотр и понимание.

Чтобы понять следующее обсуждение, будет лучше, если сайт MovieGEEKs будет работать на вашем компьютере. Вы можете зайти на него с GitHub по ссылке **mng.bz/04k5** (см. инструкции по установке на сайте).

Получение данных

Для расчета неявных оценок для конкретного пользователя вам нужно получить данные журнала пользователя. Вам нужны данные, которые должны для каждого элемента сказать вам, сколько раз пользователь взаимодействовал с контентом. Для примера с Джаспером вам нужна строка, как в табл. 4.5.

Таблица 4.5. Агрегированный вид сессий, ведущих к покупке «Эйса Вентуры» Джаспером

Идентификатор пользователя	Идентификатор контента	Подробности	Расширенные подробности	Покупка
400002	«Эйс Вентура»	3	1	1

Имея много таких строк данных, вы сможете вычислить неявную оценку. Получение таких данных может быть реализовано либо путем извлечения всех данных из журнала, содержащего идентификатор пользователя и идентификатор контента, либо вы можете заставить базу данных поработать чуть больше и возвращать данные в формате, показанном в табл. 4.5, что делается с помощью запроса SQL, показанного в следующем листинге.

ЛИСТИНГ 4.1. Код SQL для получения оценок пользователя

```
SELECT
    user_id,
    content_id,
    mov.title,
    count(case when event = 'buy' then 1 end) as buys,
    count(case when event = 'details' then 1 end) as details,
    count(case when event = 'moredetails' then 1 end) as moredetails
FROM   evidenceCollector_log log
JOIN   movies mov
ON     log.content_id = mov.id
WHERE  user_id = '4005'
group by user_id, content_id, mov.title
order by buys desc, details desc, moredetails desc
```

Р счет того, к к много копий элемент купил пользов тель

Подсчет того, сколько р з пользов в тель просм трив л подробности элемент

Подсчет того, сколько р з пользов тель просм трив л р сширенные подробности элемент

Соединение с т блицей фильм для получения з головк

Группиров ние для включения счет

Сортировк по в жности событий

Фильтр ция идентифик тор пользов теля для получения д нных одного пользов теля. Здесь это пользов тель 4005

Ср внение идентифик тор фильм с идентифик тором контент в д нных

Имея эти данные в руках (ну… в системной памяти), вы можете начать вычисление неявных оценок, которые затем должны быть сохранены в базе данных оценок. Функция реализована как показано выше и может быть найдена в коде листинга 4.2. Вы можете также посмотреть в файле *moviegeek/Builder/ImplicitRatingsCalculator.py*, чтобы найти метод `query_aggregated_log_data_for_user` и увидеть фактический код, который извлекает данные.

Расчет неявных оценок

Расчеты достаточно просты и не требуют большого числа объяснений. Вы загружаете данные из базы данных и рассчитываете оценку. Оценка рассчитывается с использованием весов, которые вы вывели ранее, как показано в следующем листинге.

ЛИСТИНГ 4.2. Расчет неявных событий по событиям пользователя

```
def calculate_implicit_ratings_for_user(userid):
data = query_aggregated_log_data_for_user(userid)
agg_data = dict()
maxrating = 0

for row in data:
  content_id = str(row['content_id'])
  if content_id not in agg_data .keys():
    agg_data[content_id] = defaultdict(int)
    agg_data[content_id][row['event']] = row['count']

  ratings = dict()
  for k, v in agg_data .items():

    rating = w1 * v['buy'] + w2 * v['details'] + w3 * v['moredetails']
    maxrating = max(maxrating, rating)
        ratings[k] = rating

    for content_id in ratings.keys():
        ratings[content_id] = 10 * ratings[content_id] / maxrating
  return ratings
```

Вызов метод , который дел ет з прос в б зе д нных

Созд ние слов ря для хр нения оценок

Проход по всем элемент м контент

Р счет неявной оценки для элемент контент

Отслежив ние высшей оценки

Возвр щение оценок

Проход по всем оценк м, норм лиз ция и умножение н 10, чтобы получить шк лу от 0 до 10

Метод `query_aggregated_log_data_for_user` вызывается для каждого пользователя. Набор данных MovieTweetings, который вы будете использовать в этой книге, включает оценки по шкале от 1 до 10, так что вы будете нормализовать эти оценки в тот же масштаб. Неявные оценки в этой шкале будут означать, что вы можете использовать их вместо явных оценок.

Отображение результата

Если у вас запущено приложение MovieGEEKs, вы можете запустить:

- `python populate_logs.py` – добавляет автоматически генерируемые журналы в базу данных. Вернитесь к главе 3 для получения дополнительной информации по этому сценарию. Теперь база данных содержит данные, показанные на рис. 4.12;

- `python -m builder.implicit_ratings_calculator` – расчет неявной оценки;
- `python manager.py runserver 8001` – перевод MovieGEEKs переключается на порт 8001.

id	created	user_id	content_id	event	session_id
100296	2017-08-14 22:04:06+02	400005	1355644	addToList	441008
100344	2017-08-14 22:04:06+02	400005	1355644	details	441008
100363	2017-08-14 22:04:06+02	400005	1355644	details	441008
100440	2017-08-14 22:04:06+02	400005	1355644	details	441009
100767	2017-08-14 22:04:06+02	400005	1355644	genreView	441014
100831	2017-08-14 22:04:06+02	400005	1355644	details	441014
100992	2017-08-14 22:04:06+02	400005	1355644	details	441019

Рис. 4.12. Фрагмент автоматически сгенерированных данных. Это данные, связанные с пользователем 400005 (Петро) и элементом 1355644

На скриншоте на рис. 4.13, вы увидите, что два фильма получили высшие оценки: 10/10. Если вы посмотрите на базу данных (рис. 4.12), вы увидите, что нет ни одного события покупки, но товар все же получил высшую оценку. Это происходит из-за многочисленных взаимодействий, которые произошли между пользователем и контентом.

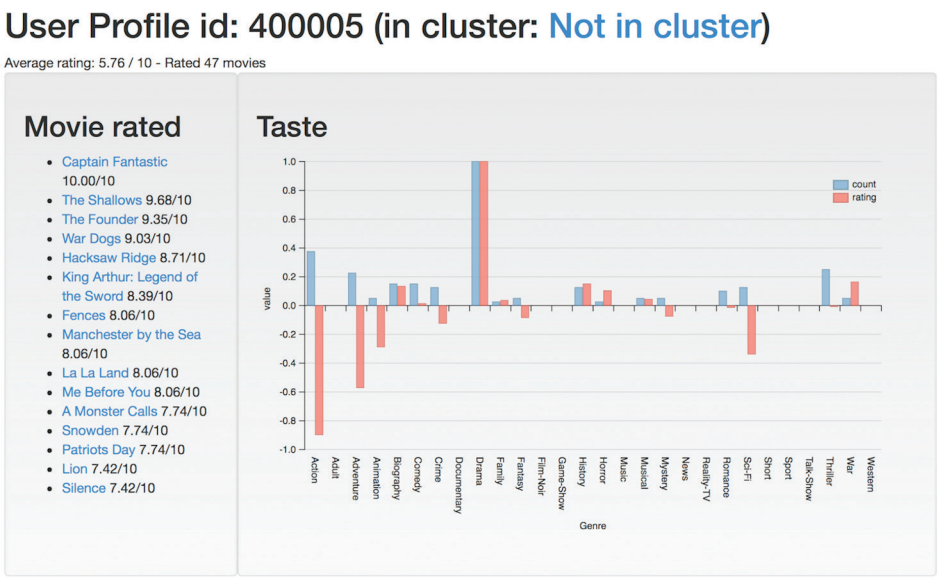


Рис. 4.13. Скриншот профиля пользователя для идентификатора пользователя 400005 (Петро). «Не в кластере» означает, что пользователь не является частью какого-либо кластера. Кластеры мы будем составлять в главе 7

При использовании неявных оценок, как те, которые описаны здесь, вы должны принять во внимание тот факт, что пользователь не потреблял многие элементы, для которых стоят оценки. В результате некупленные элементы могут быть включены в рекомендации. Но так как пользователь, который смотрел элемент, до сих пор не купил его, вы, вероятно, не будете включать его в рекомендациях.

4.6.1. Добавление учета времени

Предшествующая реализация не учитывает времени затухания. Если старое взаимодействие менее важно, чем недавнее, стоит добавить его в смесь. Учесть временной аспект стоит немного больших усилий, потому что вы должны посмотреть на каждую точку данных и добавить множитель, который будет становиться меньше с течением времени.

Добавление времени затухания может быть сделано с помощью SQL в базе данных или с помощью кода. Некоторые компании говорят, что делают все в SQL, а другие говорят, что SQL становится нечитаемой с более чем 10 строк, поэтому делают все в коде. Функция времени затухания, которую мы собираемся реализовать, использует следующую формулу:

$$\text{вес} = \frac{1}{\text{затухание в днях}}.$$

В базе данных SQL нет элегантного способа сделать это, так что мы реализуем затухание в коде. Это также дает возможность увидеть, как вы можете сделать все в коде, в отличие от листинга 4.1, в котором агрегируются данные. Здесь вы получите все данные журнала от пользователя с помощью запроса в следующем листинге[1].

ЛИСТИНГ 4.3. Код SQL для возвращения данных журнала пользователя

```
SELECT *
FROM collector_log log
WHERE user_id = {}
```

Есть две причины использования дней вместо секунд. Во-первых, вы хотите, чтобы все события, которые происходили в один день, считались одним и тем же, потому что фильмы не покупаются по несколько раз в день. Во-вторых, вы хотите реализовать медленное затухание. Если мы используем дни, вес событий, произошедших за неделю, уменьшится только на одну седьмую. Музыкальные стриминговые сайты вроде Spotify, возможно, уделяют большее внимание последнему часу или даже последним 10 мин. На рис. 4.14 показан алгоритм затухания, на котором вы можете увидеть график времени затухания.

[1] Если файл журнала слишком велик, хорошим решением будет ограничить его ведение по времени, например «Создано более месяца назад», и определить указатель в `user_id` столбца

Чтобы убедиться, что вы понимаете происходящее, давайте рассмотрим пример. Если Джимми сегодня купил «Игру престолов», событие будет иметь 1 балл, если покупка была вчера – то 1/2, неделю назад – 1/7, год назад – 1/365. Функция затухания показана в следующем листинге.

ЛИСТИНГ 4.4. Вычисление неявных оценок с использованием событий пользователя и времени затухания

```
def calculate_implicit_ratings_w_timedecay(userid, conn):

    data = query_log_data_for_user(userid, conn)

    weights = {{'buy': w1}, {'moredetails': w2}, {'details': w3} }

    ratings = dict()
    for entry in data:
       movie_id = entry.movie_id
       event_type = entry.event

    if movie_id in ratings:

        age = (date.today()-entry.created)//timedelta(days=365.2425)

        decay = calculate_decay(age)

        ratings[movie_id] += weights[event_type]*decay

return ratings
```

З хв т д нных

Слов рь весов для к ждого тип событий

Созд ние пустого слов ря для оценок

Пок з внутреннего з прос и доб в- ление времени з тух ния

Р счет возр ст события

Получение коэффициент з тух ния

Приб вление вес события, умноженного н коэффициент з тух ния

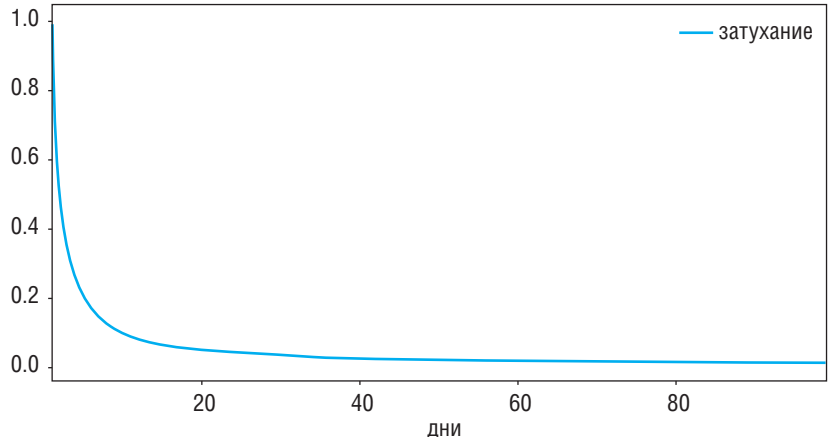


Рис. 4.14. Функция алгоритма затухания

Функция затухания показана в следующем листинге.

ЛИСТИНГ 4.5. Функция затухания

```
def calculate_decay(age_in_days):
    return 1/age_in_days
```

Вы можете попробовать более сложные алгоритмы затухания, чтобы увидеть, получится ли что-то улучшить. На данный момент вам, вероятно, трудно понять, какой эффект получится от изменения функции затухания. Суть в том, что мы добавляем указание на то, насколько старые вещи релевантны для пользователей.

Если у вас есть лошадь, вы, вероятно, не собираетесь часто менять предпочтения в оснащении для лошади, но если вы посмотрите новостной сайт, то, скорее всего, старые истории вам будут неинтересны. Для фильмов один день – это короткий период, и один месяц (и даже год) тоже может быть релевантным. Для тестирования системы, однако, имеет смысл использовать один день.

4.7. Более редкие элементы имеют бóльшую ценность

Популярные элементы часто (по понятным причинам) предлагаются людям. Но если вы хотите понять, что людям важно, нужно посмотреть, что пользователи покупают. Рассмотрим следующие примеры:

- я покупаю фильм «Властелин колец», который по графикам был лучший, и каждый человек и его собака уже купили его;
- я покупаю расширенную версию специального коллекционного издания «Властелина колец», куда входит подписанный актерами постер и которого существует всего 100 экземпляров.

Какое событие говорит о моих вкусах? Первый пример ставит меня в один ряд с половиной земного шара, и большинство людей в первом примере не делают меня уникальным по сравнению с другими. Второй пример делает меня уникальным, ведь в моем клубе по интересам всего 100 человек. И эти 100 человек имеют специфический вкус, который, вероятно, будет для них общим.

Другой пример, который часто используется для описания этой проблемы, – это бананы. Все покупают бананы, поэтому знание о том, что пользователь покупает бананы, не имеет большого значения, особенно по сравнению с импортируемыми сардинами в масле чили – редкий продукт, который кое-что говорит о покупателе. Почему я упоминаю об этом? Ну, вы могли бы поставить фильтр на оценки и «приподнять» редкие элементы по сравнению с обычными. Реализация этого также может быть немного сложнее, так что давайте быстренько пройдемся по ней.

Во-первых, давайте определим функцию. Эта проблема тесно связана с хорошо известной проблемой *частоты термина–обратной частоты документа* (TF-IDF). Она часто используется поисковыми системами в качестве инстру-

мента для ранжирования значения сводного документа при заданном запросе пользователя. Вы можете считать это беззапросным поиском, в котором вы хотите увеличить ценность особенных элементов. Чтобы понять, какие элементы контента считать особенными, вам понадобится IDF. Идея заключается в том, что, если пользователь покупает популярный элемент, это дает мало информации о вкусе пользователя. Если же пользователю нравится что-то особенное, это может быть лучшим показателем личного вкуса.

Эта функция может быть вычислена несколькими способами. Я покажу более ценный[1]. Чтобы найти особенные элементы, вы можете вычислить обратную частоту пользователя (*iuf*):

$$iuf_{i,u} = \log\left(\frac{N}{1+n}\right),$$

где:

- n – это количество раз, сколько элемент i был куплен пользователем u;
- N – количество пользователей в каталоге.

Нормализация в этом контексте означает, что мы откладываем результат на логарифмической шкале. Это делается, чтобы лучше увидеть большую разницу между маленькими числами и наоборот. Вы можете увидеть это на рис. 4.15.

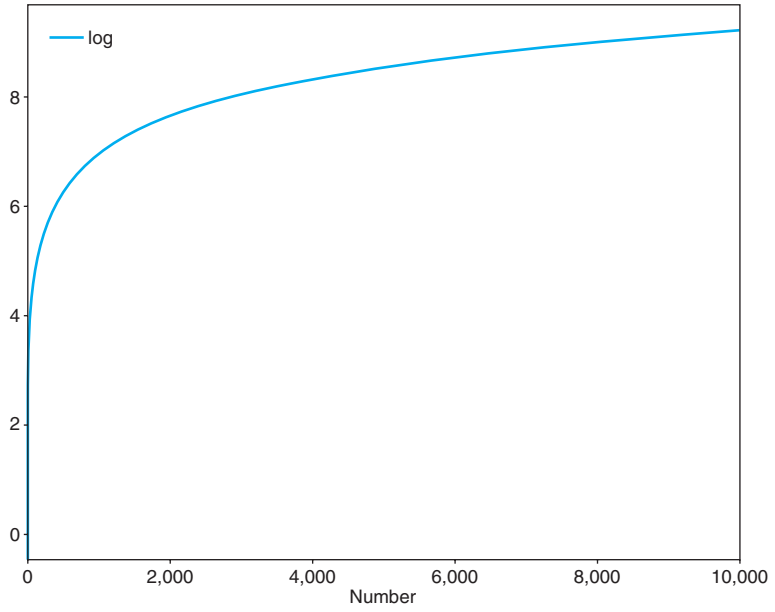


Рис. 4.15. Логарифмическая функция для чисел от 1 до 10000. Она используется, чтобы лучше увидеть большую разницу между маленькими числами и наоборот.

[1] Он упоминается в связи с совместной фильтрации в статье Дж.С. Бриза и др. «Эмпирический анализ предсказательных алгоритмов совместной фильтрации», которую я рекомендую вам прочитать. См. **mng.bz/tQhY**.

Поскольку число пользователей N при расчете IUF постоянно, этот график будет выглядеть так, как показано на рис. 4.16. Взвешенная оценка вычисляется путем умножения следующим образом:

$$wR_{i,u} = R_{i,u} \times iuf_{i,u} = R_{i,u} \times \log\left(\frac{N}{1+n}\right).$$

Если вы хотите сохранить оценки в диапазоне от 1 до 10, вам нужно нормализовать их, как мы делали в листинге 4.2. Это позволяет легче судить об этих оценках и сравнивать их с явными. Если вы считаете, что ваш сайт может увеличить количество переходов, это может быть реализовано в SQL и на сервере.

Помните, что если полученные рекомендации согласуются с данными, но не с пользователями, то настраивать нужно именно расчет неявных оценок. Неявные оценки, которые вы рассчитываете, – это фундамент для ваших рекомендаций.

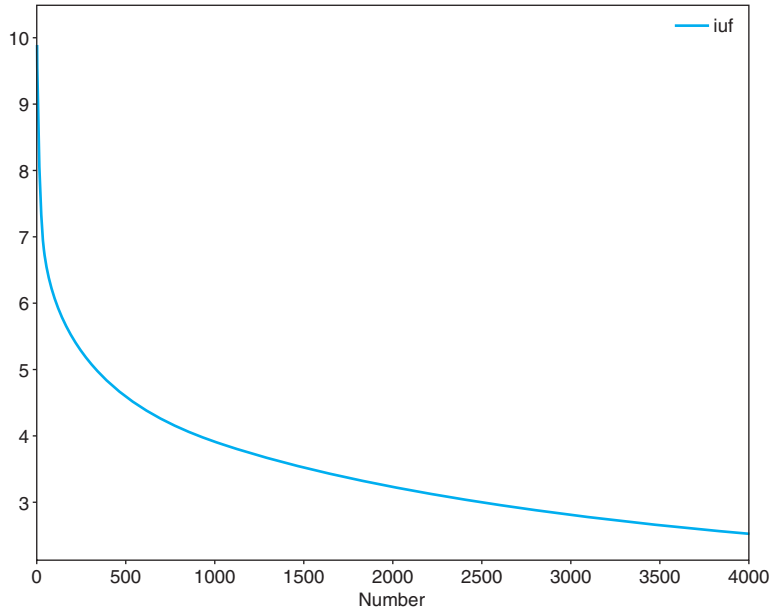


Рис. 4.16. IUF выглядит следующим образом. Если всего несколько человек купило продукт, значение будет больше, и наоборот

Если вы сделаете это хорошо, рекомендатор получит лучшие условия для предсказаний, а если плохо – рекомендатор потерпит неудачу.

Теперь, когда у вас есть панель и неявные оценки, вы готовы к созданию рекомендательной системы. Следующим шагом будет то, как лучше всего сохранить объединение пользователя, контента и данных об оценках. Это часть следующей главы, где вы также увидите несколько рекомендаций. Вперед!

Резюме

- Матрица пользователь–элемент является форматом данных для рекомендательных алгоритмов. Вы можете заполнить ее явными и неявными оценками или путем указания того, какие элементы были потреблены пользователем в бинарной матрице.
- Оценка – это клей, который соединяет пользователя с элементом. Она либо может быть введена вручную пользователем, либо рассчитывается на основе поведения пользователя.
- Алгоритм времени затухания учитывает, что не вся информация в равной степени важна: старые доказательства менее важны, потому что люди имеют тенденцию изменять свои вкусы.
- Следует использовать обратную частоту, поскольку взаимодействие с менее популярными элементами дает более подробную информацию о пользователе, который взаимодействует с популярными элементами.

Глава 5

Неперсонализированные рекомендации

В этой главе мы поговорим о рекомендациях, но они не персонализированы. Однако это не означает, что глава будет менее важной:

- вы узнаете, что использование неперсонализированных рекомендаций может показать интересный контент;
- вы увидите примеры, показывающие, почему нужно упорядочивать контент, и узнаете, как строить диаграммы, которые показывают популярные элементы и выделяют элементы, представляющие интерес для других пользователей;
- вы узнаете, как рассчитать ассоциативные правила, создавая множества элементов на основе покупательской корзины, а затем использовать эти правила для создания рекомендаций;
- вы увидите, как реализован рекомендатор на примере сайта MovieGEEKs.

Неперсонализированные рекомендации – это, как правило, то, с чего большинство сайтов начинают, потому что реализовать их легко и не требуется знать что-либо конкретное о пользователях. Неперсонализированные рекомендации хороши, потому что вы всегда можете показать их, несмотря на то, как мало вы знаете о пользователях. Кто-то может сказать, что такие рекомендации годятся только до тех пор, пока система мало знает о пользователе. Но помните, что люди подобны стаду, и многим достаточно будет просто самых популярных элементов.

В этой главе мы будем обрабатывать графики и составлять ассоциативные правила. Сперва посмотрим на старые добрые графики, которые все ненавидят в нашу эпоху контекста. Графики – это простые рекомендации, основанные на статистических данных, например о том, какого контента было продано больше. Графики позволяют упорядочить данные, и именно с этого мы начнем разговор. Мы посмотрим на реализацию графиков и обсудим упорядочение фильмов на сайте MovieGEEKs. Во второй части этой главы мы рассмотрим,

что люди кладут в корзину, и создадим рекомендации типа «люди, которые купили *X*, также купили *Y*», используя так называемые *множества элементов* или же *частотные множества*. Рекомендации, которые мы рассмотрим, будут одинаковыми для всех пользователей, взаимодействующих с системой; поэтому они называются неперсонализированными рекомендациями.

После четырех глав, которые в основном были посвящены тому, как собирать данные о пользователях, наверное, немного несправедливо забыть про персональность и смотреть на сырые данные. Но помните, что у большинства сайтов много неопознанных посетителей, которых тоже надо удовлетворить, потому что они – будущие клиенты вашего сайта. И даже если вы знаете личность вашего посетителя, весьма вероятно, что у вас не будет достаточного количества данных для расчета персонализированных рекомендаций, поэтому можно обойтись неперсонализированными.

5.1. Что такое неперсонализированные рекомендации?

В главе 1 мы обсуждали разницу между рекламой и рекомендацией. Давайте еще раз кратко поговорим об этом.

5.1.1. Что такое реклама?

Сделка дня на сайте Manning, показанная на рис. 5.1, является рекламой. Это не делает ее плохой, ведь реклама – это то, что продавец хочет показать пользователям, и иногда пользователям она интересна.

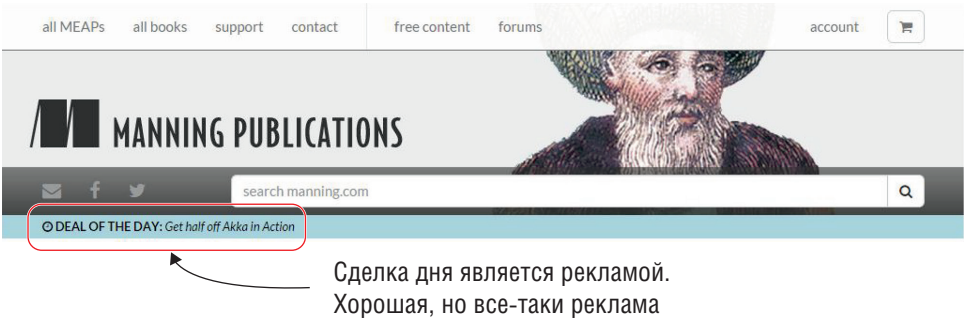


Рис. 5.1. Сделка дня на сайте Manning Publications

Но с этим вы, как владелец сайта, должны обращаться с осторожностью, потому что плохая реклама оставит вас без посетителей. (Я думал, в интернете полно такой рекламы, но после 30 мин поиска сдался. Я нашел жалобу от парня, который получал кучу спама от сайта с паранормальными свиданиями, но он сам виноват, что считал его серьезным.) Я определяю плохую рекламу как что-то, что мешает пользователю делать свои дела, как, например, всплывающее окно, которое нельзя закрыть, или принудительные переходы.

Часто *реклама* – это попытка продавца убедить пользователя в том, что у него есть что-то хорошее или дешевое (даже если это не так), в то время как *рекомендация* – это то, чего хочет сам пользователь. Можно сказать, что именно подешевле-то и хочет пользователь. На самом деле многие сайты сделали бизнес, рекомендуя дешевые вещи. Лично я иду искать что-то, что мне нужно, в то время как сайты с купонами типа cupon.com (рис. 5.2) предназначены для людей, которые не знают, чего хотят.

На cupon.com используются неперсонализированные рекомендации. В верхней части перечислены самые популярные категории и бренды. В центральной части экрана содержится список ваучеров, позволяющих сэкономить деньги. Cupon.com является одним из многих таких сайтов, и я думаю, что это отличный способ для продавцов найти людей, которые любят «подешевле», но таким образом тратят еще больше за счет количества.

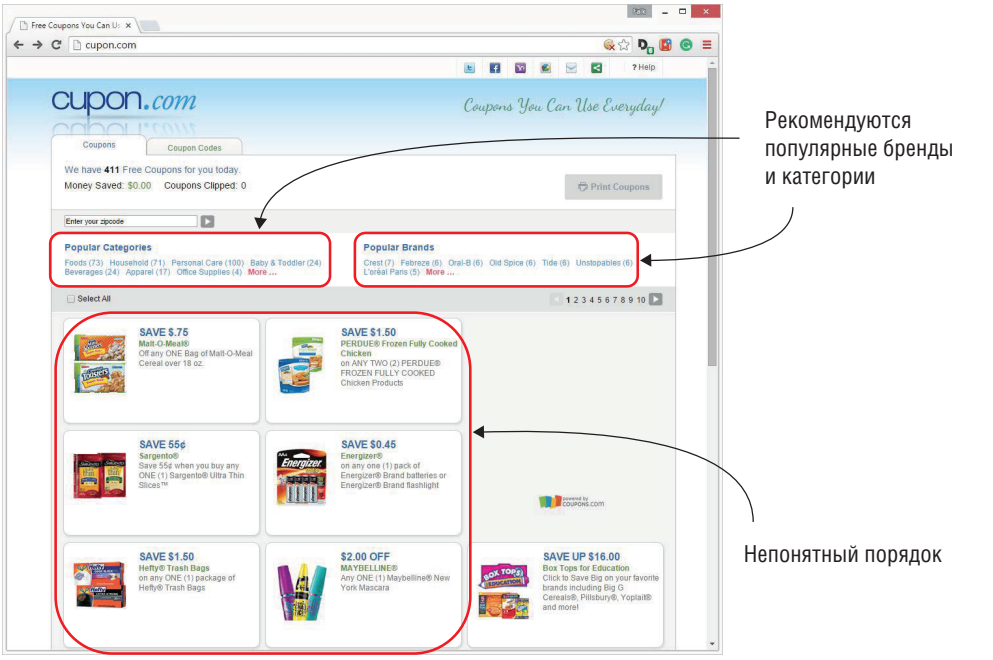


Рис. 5.2. Cupon.com собирает купоны отовсюду

5.1.2. Что делает рекомендация?

Рекомендация, персонализированная или нет, основывается на данных и рассчитывается из них. Чтобы это определение не было слишком мутным, мы ограничим его компьютерным расчетом данных. Это означает, что популярные категории в cupon.com являются плодом рекомендаций (по категориям). Перед тем как начать расчет, давайте посмотрим на примеры того, что сайт может сделать, если у него нет никаких данных.

5.2. Как сделать рекомендации, когда у вас нет данных

Мы говорили об этом раньше – отсутствие данных вообще не означает отсутствие рекомендаций. Рекомендации происходят из того, что нравится людям. Опять же, отсутствие данных не означает отсутствие рекомендаций. Что вы можете сделать? Для начала можете подделать рекомендации с помощью кода вручную (хотя это уже будет почти реклама). Что-то подобное делает DZone.com, как показано на рис. 5.3. Если вы не хотите (или не имеете возможности) делать такую страницу, можете сделать уменьшенную версию.

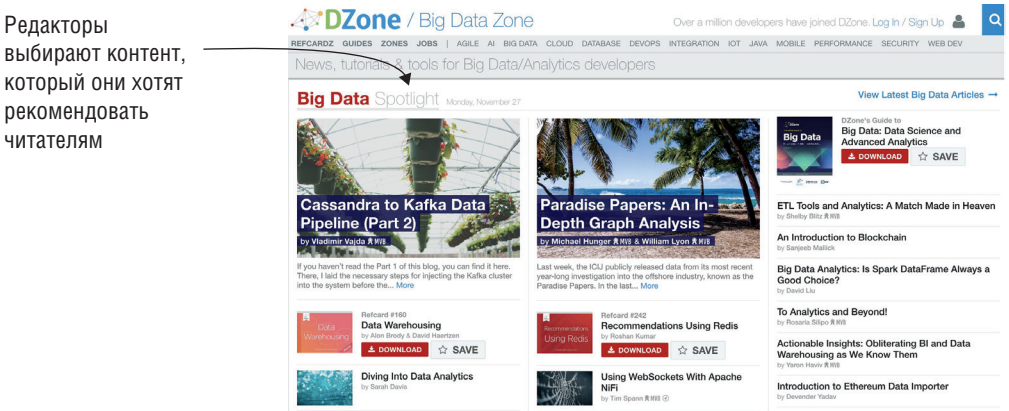


Рис. 5.3. Созданные редакторами рекомендации в DZone

Во-первых, что значит представить неупорядоченный данные? Примером этого является страница на MovieGEEKs, где фильмы, показанные на первой странице, – это первые элементы, которые вышли из базы данных. Но представление данных таким образом устраняет возможные совпадения. Данные могут быть включены в базу данных в алфавитном порядке, или если вы всегда добавляете данные в конец, то самые старые данные будут отображаться первыми.

> **ПОДСКАЗКА.** Никогда не используйте порядок, заданный в базе данных. Всегда стоит рассматривать возможность упорядочения контента по умолчанию.

Сортировка по цене – обычно плохо

Перед сортировкой данных по цене следует подумать, что это значит. В случае с cupon.com это может означать, что элементы с наименьшей выгодой будут сверху или наоборот. Другой вариант – сортировка по проценту экономии. Например, если вы экономите 1 долл. из товара, который стоит 10 долл., этот элемент даст экономию больше, чем 2 долл. из 100. В первом случае скидка составляет 10 %, а в другом – 2 %.

Недавние данные позволяют сохранить динамику сайта

На сайте с фильмами нет цен – и что с этим делать? Для начала одним из самых простых способов может быть сортировка по признаку того, что людям, скорее всего, понравится (если вы знаете, что это). Вы можете отсортировать фильмы по дате выпуска. В случае с фильмами, вероятно, самые новые должны отображаться в первую очередь. В этом случае сайт оживет и станет более динамичным, пока контент тоже живой и динамичный.

Помните, что самые новые элементы не всегда наиболее желаемы. Если вы продаете антиквариат, вероятно, лучше сначала выводить самые старые, как на сайте PreWarCar.com (**www.prewarcar.com**), – см. рис. 5.4.

Или другой пример – если ваш элемент является садовым инструментом, то людям не интересна новизна, если это не грили Weber из Дании.

PreWarCars является одним из немногих примеров, когда самые старые лучше выводить вверх

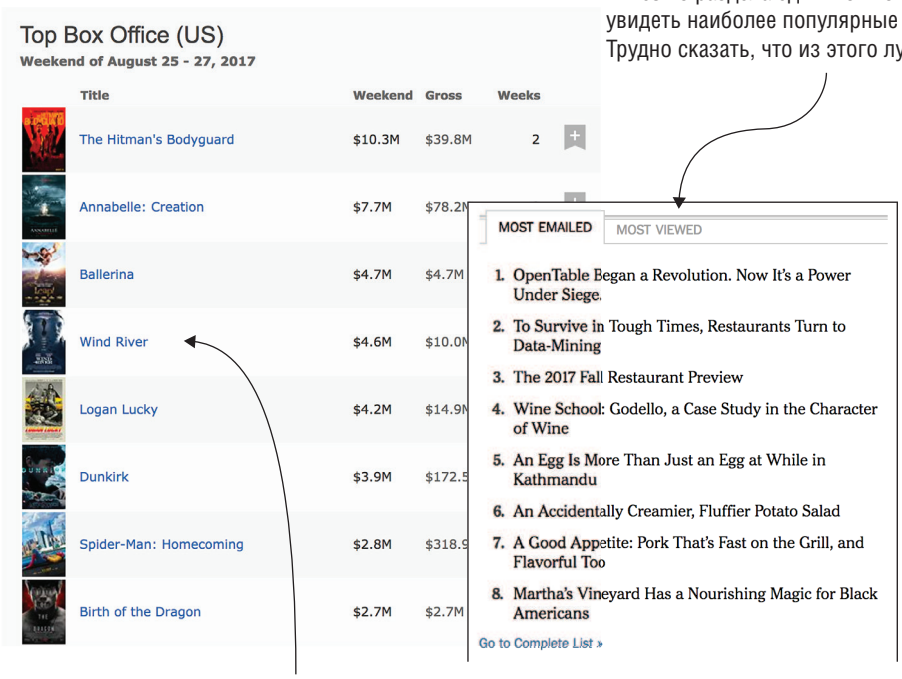
Рис. 5.4. PreWarCar.com, площадка по продаже антикварных автомобилей

Если вы мужчина датчанин, вы, наверное, покупаете гриль Weber по крайней мере, один раз в год, а может и больше. А все благодаря гениальному маркетингу.

5.2.1. Топ-10: диаграмма элементов

Еще до интернета была мода на всякие топ-10. На радио каждые выходные крутили чарты топ-10, которые мы свято записывали и слушали на повторе всю неделю. Топ-10 был, по сути, единственный способ получить рекомендации, помимо мнения друзей. Когда MTV приехали в Данию, это было безумием …

Во всяком случае эти топы, как показано на рис. 5.5, получили плохую репутацию, так как показывали популярные элементы независимо от вашего вкуса.



Топ-10 рассылок статей по электронной почте The New York Times из раздела еды. Можно также увидеть наиболее популярные статьи. Трудно сказать, что из этого лучше

Фильмы с самыми большими кассовыми сборами за выходные в 2017 году в США. Не знаю, почему прибыль является хорошим показателем качества, но часто сортируют по ней

Рис. 5.5. Топ-10 фильмов по кассовым собрам по версии iMDB (imdb.com) и рассылок о еде издания The New York Times (nytimes.com)

Список топ-10 – это все, что вам нужно, не так ли? А вот и нет. Такая диаграмма способна удовлетворить даже большинство, но все пользователи разные, поэтому рекомендатор даст все же лучший результат. Но и топ-10 тоже несет информацию. Если у вас есть, скажем, 11 пользователей и 10 фильмов, то самый популярный фильм понравится двум людям, а девять других могут любить что-то еще. Всегда нужно учитывать такие данные.

Давайте посмотрим на то, как может выглядеть реализация графика. Диаграмма показывает, какие элементы контента покупались наиболее часто, например на рис. 5.6. Мы также могли бы сделать диаграмму, на которой элементы просматривались чаще или, используя термин Facebook, понравились больше.

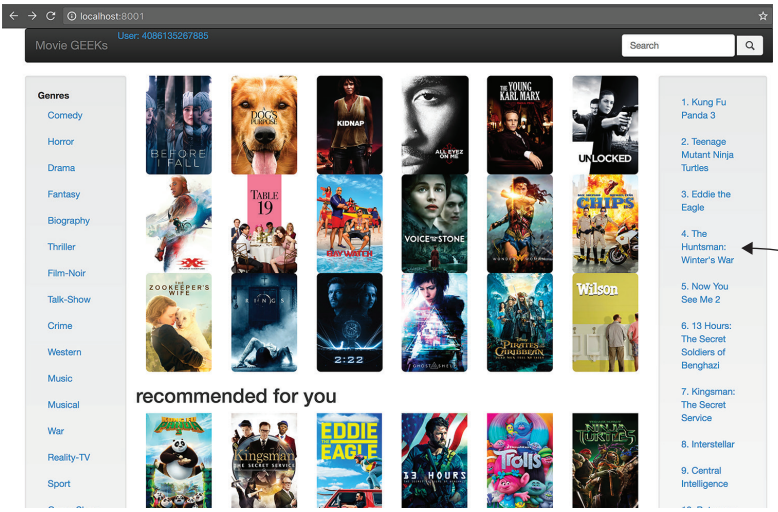


Рис. 5.6. MovieGEEKS выводит диаграмму, показывающую, что наиболее часто покупают

5.3. Реализация диаграмм и основы для рекомендатора

Можно легко добавить на сайт диаграмму, об этом мы говорили в главе 4. Для того чтобы сделать все правильно и добавить функциональность в нужном месте, мы начнем с создания компонента рекомендательной системы, и именно им мы будем заниматься большую часть главы.

5.3.1. Компонент рекомендательной системы

Рекомендаторная система может быть построена по-разному в зависимости от того, сколько рекомендаций нужно вывести, от объемов каталога контента и количества посетителей. Точно могу сказать, что вы хотите, чтобы это была не зависящая от вашего сайта структура, иначе у вас «просядет» производительность.

Решение для сайта MovieGEEKs состоит из двух компонентов (и базы данных). Первый компонент, *конструктор*, выполняет все предварительные расчеты (обучение), необходимые для вывода рекомендаций, в то время как второй компонент ориентирован на рекомендации. Причина в необходимости компонента *конструктора* заключается в том, что для большинства рекомендаций нужно много вычислений, а это требует времени, которого у вас нет, ведь пользователь ждет загрузки страницы. Цель большинства рекомендательных алгоритмов – вычислить как можно больше, чтобы сделать работу в режиме реального времени как можно быстрее.

Как обычно, рекомендательные алгоритмы разделяются на алгоритмы на основе памяти и на основе модели. «На основе памяти» означает, что рекомендатор получает доступ к данным журнала в режиме реального времени,

а «на основе модели» – что алгоритм агрегирует данные заранее, что делает его более отзывчивым. Опыт показывает, что алгоритмы на основе памяти работают только до определенной точки, поскольку при слишком большом числе просмотров поддерживать работу сервера трудно.

Посмотрите на рис. 5.7 и скажите, что бы вы сделали и почему? В светлых квадратиках указаны компоненты, которые мы уже обсуждали. Темные квадратики – это тема данного раздела. API рекомендатора обрабатывает запросы с веб-сайта. База данных рекомендаций содержит расчетные рекомендации. Конструктор рекомендаций – это компонент, который создает модель и в некоторых из алгоритмов выполняет расчет рекомендаций и сохраняет их в базе данных.

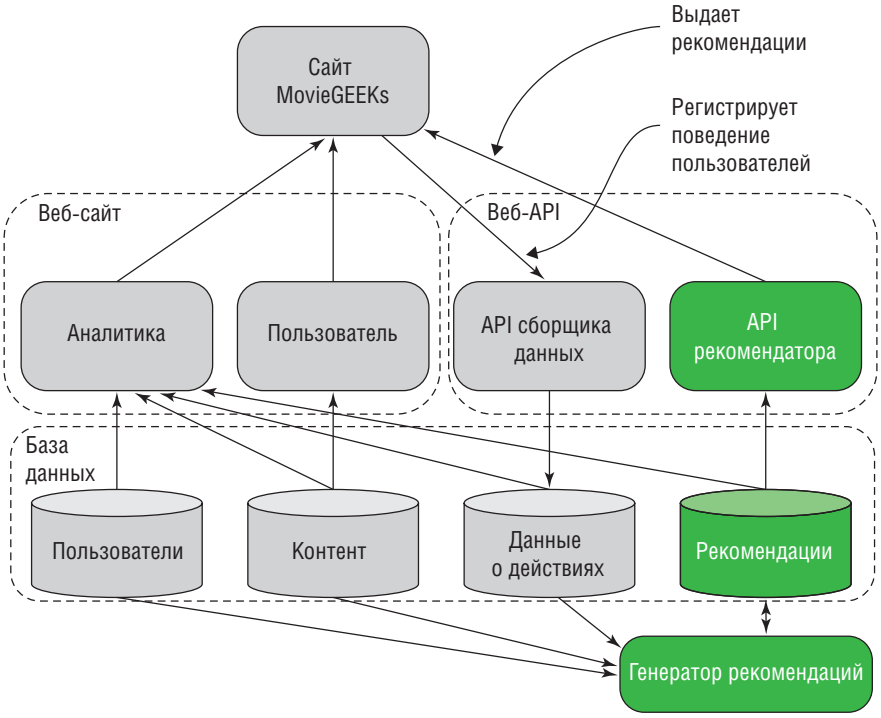


Рис. 5.7. Архитектура MovieGEEKs с выделенной рекомендательной системой

5.3.2. Код MovieGEEKs на сайте GitHub

Повторим, что вам стоит загрузить код сайта MovieGEEKs с сайта GitHub по ссылке **mng.bz/04k5**. Установите его, следуя инструкциям в файле инструкций в корневом каталоге, и поэкспериментируйте с ним.

5.3.3. Рекомендательная система

Рекомендательная система – это приложение, которому требуется доступ ко всем вашим данным в зависимости от того, какого типа рекомендации нужно производить. Часто у сайтов рекомендатор обособлен от остального сайта, и на всякий случай используется резервный рекомендатор, который будет продолжать предоставлять данные, если рекомендательная система «сломается».

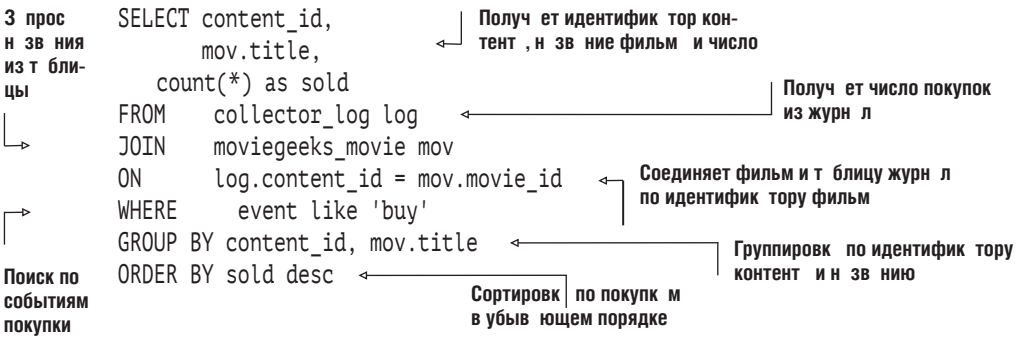
> **ПРИМЕЧАНИЕ.** Всегда храните рекомендательную систему отдельно от других частей вашего сайта, потому что она весьма требовательна с точки зрения производительности.

На сайте MovieGEEK рекомендательная система реализована в виде отдельного приложения на Django. Это позволяет ему работать независимо и на отдельной машине, когда сайт выходит в производство.

5.3.4. Добавление диаграмм на MovieGEEKs

Диаграмма показывает довольно простую вещь; вам нужно подсчитать, сколько раз был куплен каждый элемент, и отложить на графике эту информацию. При наличии SQL это делается путем группировки по контенту и подсчета числа покупок в журнале событий. SQL-запрос в листинге 5.1 делает еще одну вещь: он добавляет на диаграмму название фильма. В живой системе, возможно, стоит рассчитывать график один раз в день, потому что таблицы довольно велики, а делать запрос каждый раз накладно с точки зрения производительности.

ЛИСТИНГ 5.1. SQL-запрос, позволяющий вывести наиболее продаваемые элементы



Вы запрашиваете файл журнала, который будет пустым, если вы недавно загрузили код. Но если вы запустите сценарий *moviegeek/populate_logs.py* в корневом каталоге, он сгенерирует для вас данные журнала. Этот сценарий более подробно описан в главе 4.

Оставаясь в духе рекомендаторов, скажем, что это рекомендация, и поэтому должно быть что-то, что должно выйти из приложения рекомендатора. Метод графика в *recommender/views.py* показан в листинге 5.2. Это не очень хороший код, но он работает.

ЛИСТИНГ 5.2. Метод графика в файле recommender/views.py

```
def chart(request, take=10):
    sorted_items = PopularityBasedRecs().recommend_items_from_log(take)
    ids = [i['content_id'] for i in sorted_items]

    ms = {m['movie_id']: m['title'] for m in
        Movie.objects.filter(movie_id__in=ids).values('title',
                                                      'movie_id')}
    sorted_items = [{'movie_id': i['content_id'],
                     'title': ms[i['content_id']]} \
                    for i in sorted_items]
    data = { 'data': sorted_items }

    return JsonResponse(data, safe=False)
```

Вызов рекоменд тор по популярности с целью получить н иболее покуп емые элементы

Извлек ет идентифик торы фильмов в список

Использует идентифик торы фильмов для получения з головков фильмов

Возвр т json-код

Созд ние нового сортиров нного списк элементов с н зв ниями фильмов

Метод, используемый в предыдущем листинге, вызывает следующий метод для извлечения данных из журнала.

ЛИСТИНГ 5.3. Метод recommend_items_from_log в файле recommender/views.py

```
recs/popularity_recommender.py
def recommend_items_from_log(self, num=6):
    items =
     Log.objects.values('content_id')
    items = items.filter(event='buy').annotate(Count('user_id'))

    sorted_items = sorted(items, key=lambda item: -float(item['user_id__count']))

    return sorted_items[:num]
```

Извлек ет элементы и подсчитыв ет, к к ч сто они покуп ются

Сортировк по числу покупок

Чтобы увидеть результат работы метода, введите URL-адрес **localhost:8000/recs/chart** в адресной строке браузера. У меня выводится диаграмма 10 самых продаваемых товаров для MovieGEEKs, показанная ранее на рис. 5.6.

5.3.5. Заставим контент выглядеть более привлекательно

Давайте посмотрим на контент MovieGEEKs (фильмы в базе данных). Ранее мы говорили, что вы не можете позволить базе данных диктовать порядок того, как вы представляете свой контент. Если сайт MovieGEEKs будет показывать фильмы так, как они есть в базе данных, он будет выводить очень старые фильмы (рис. 5.8).

Рис. 5.8. Фильмы как они есть в базе данных

Вы можете поиграть с сортировкой базы данных, изменив эту строку в файле на следующую:

ЛИСТИНГ 5.4. Отбор старейших фильмов

```
movies = selected.movies.order_by('year')
```

Эта команда сортирует фильмы по годам, начиная с самых старых. Если вы любите черно-белые фильмы, это здорово, но я думаю, что большинству людей интереснее что-нибудь поновее и поцветнее. Для того чтобы отобразить самые последние фильмы в запросе Django, добавьте знак «минус» перед названием столбца, как показано в этом списке.

ЛИСТИНГ 5.5. Отбор новейших фильмов

```
movies = selected.movies.order_by('-year')
```

Это изменение делает первую страницу MovieGEEKs немного интереснее, как видно на рис. 5.9.

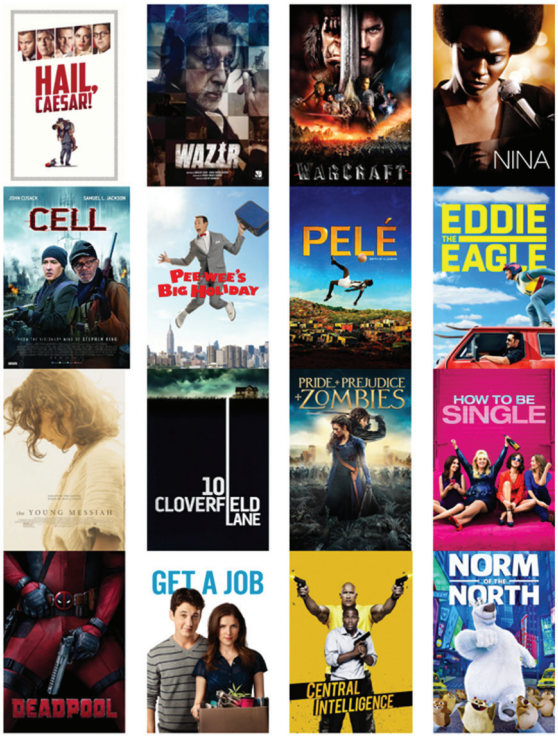


Рис. 5.9. Вывод недавно добавленных фильмов

Вы также можете отсортировать фильмы по дате выпуска, но это тоже не всегда хорошо, потому что клиенты не всегда ищут самое новое. Например, Netflix часто добавляет в свой каталог новый контент, но это не всегда недавно вышедшие фильмы, так что принцип сортировки должен быть другим.

Вернемся к садовым инструментам: от сортировки по дате производства мало толку, но, возможно, в базе данных будут данные о том, когда садовые инструменты обычно используются. Вы можете использовать машины для подготовки почвы в начале осени, затем вам нужны инструменты, чтобы посеять семена, затем что-то от сорняков и т. д. Садовое оборудование можно сортировать по сезонному использованию, выводя сперва самые актуальные. Когда сортировка выполнена и получено несколько неперсонализированных рекомендаций (рис. 5.6), настало время посмотреть более конкретные рекомендации.

5.4. Выборочные рекомендации

Графики хороши, но чересчур все обобщают. Многие сайты берут какой-то элемент и выводят в рекомендации связанные элементы. Эти элементы можно назвать *сидами*.

Не слишком ли это похоже на поиск? Похоже, но идея состоит в том, что отобранные рекомендации могут быть элементом, продуктом или позицией, которые вы затем используете в качестве входных данных, чтобы найти связанный контент. Вы можете использовать покупаемые вместе элементы, а для того чтобы понять, как люди покупают вместе, нужно изучить данные.

Одним из самых известных выборочных рекомендаций является раздел **Часто покупают вместе** (FBT) (рис. 5.10), который есть и на Amazon, и в почти любом другом интернет-магазине. Создание таких ассоциаций между элементами называется *анализом родства* или привычнее – *анализом корзины покупок*. Давайте посмотрим, как это сделать.

Рис. 5.10. Пример раздела **Часто покупают вместе** на Amazon

5.4.1. Часто покупаемые элементы, похожие на тот, который вы просматриваете

Можно ли создать FBT-рекомендатор, найдя все продукты, которые покупаются вместе с текущим продуктом, а затем взять топ-X из них? Можно, но, как вы увидите позже, это не очень хорошо работает.

Одна из проблем FBT-рекомендаций заключается в том, что большинство продуктов покупается вместе с другими популярными элементами. Классический пример: у большинства людей в супермаркете в Дании в корзине будет литр молока почти независимо от того, что еще находится в корзине. То есть связи между продуктами особо нет.

Пока я это писал, моя жена вернулась домой из супермаркета даже с двумя литрами молока (на рис. 5.11 ее чек). Большинство продуктов в супермаркете покупается вместе с молоком.

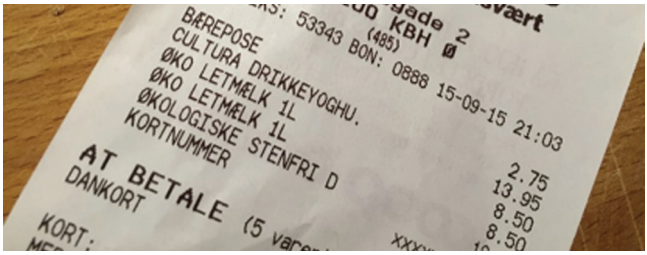


Рис. 5.11. Что купила моя жена (пакет, йогурт, молоко и оливки)

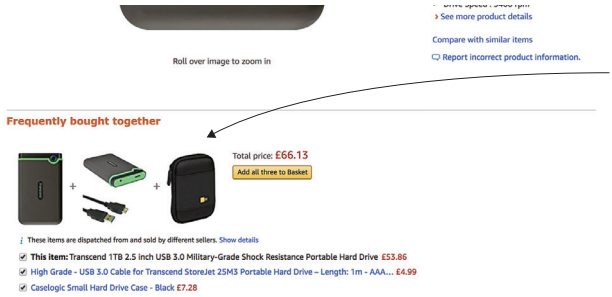
Это дает список с частотой покупки, где будет молоко и большинство других продуктов. Когда два или более элементов часто оказываются в таких списках вместе, это называется *частотным набором*.

Вы, наверное, думаете: «Да, здорово, но это интересно только для владельцев супермаркетов». Однако это вполне применимо и для многих других областей.

В следующих разделах мы рассмотрим, как рассчитать FBT-рекомендации на простом примере супермаркета, а затем перейдем к реализации этого на сайте с фильмами. Кроме того, можно перейти к более крупным предметам, например мебели или недвижимости продаж. Или, например, если сайт продает лодки, то в комплекте с ними наверняка понадобится спасательный жилет. И в зависимости от того, парусная это лодка или катер, возможно, потребуется специальное оборудование. Даже если вы продаете большие и дорогие вещи, стоит добавлять к ним FBT-рекомендации, не так ли?

5.4.2. Ассоциативные правила

Вместо того чтобы отбирать наиболее популярные элементы, можно использовать ассоциативные правила. Это хорошая вещь, несмотря на заверения маркетологов. *Ассоциативные правила* в коммерческих сценариях можно рассматривать как благонамеренные советы. Большинство людей будут недовольны, вернувшись домой с новым жестким диском и осознав, что у них нет кабеля для его подключения. Если вы покупаете вещи на Amazon, то вам повезло, потому что сайт напомнит вам, что большинство людей покупают кабель вместе с жестким диском, что и показано на рис. 5.12.



Большинство людей не умеет телепортиров ть д нные, поэтому для перед чи нужен к бель. К сч стью, Amazon будет р д н помнить в м об этом

Рис. 5.12. Amazon подсказывает, что внешние жесткие диски часто покупают вместе с кабелем

Теперь давайте немного подумаем о том, как получить эти ассоциативные правила. Ниже приведен список чеков из супермаркета. (Я пытался добавить сюда пример о «Звездных войнах», но не получилось.) Давайте представим, что у вас есть супермаркет, в котором продается всего пять продуктов: молоко, финики, йогурт, морковь и хлеб. Когда речь идет о правилах ассоциации, они, как правило, называются *элементами*.

1. {хлеб, йогурт}
2. {молоко, хлеб, морковь}
3. {хлеб, морковь}
4. {хлеб, молоко}
5. {молоко, финики, морковь}
6. {молоко, финики, йогурт, хлеб}

Каждая строка – это покупка или транзакция, в которой содержится ряд элементов. Чтобы составить правила ассоциации, посмотрите на элементы, которые приобретаются вместе. Если в данном примере вы выберете молоко, то увидите, что все остальные продукты покупаются вместе с молоком. Ценная ли это информация? Кажется, что нет. И что тогда?

Вам нужно найти элементы, которые всегда покупают вместе, но не покупаются со всем остальным. Любое подмножество элементов списка называется *множеством элементов*. Хлеб и молоко образуют множество *{хлеб, молоко}*, и такое множество присутствует в трех транзакциях из шести. Дает ли это информацию о том, что стоит рекомендовать молоко, когда есть хлеб в корзине? Да.

Давайте определим некоторые цифры, которые позволяют легче решить, появилось ли у нас правило или это просто совпадение. Проблема нашего множества в трех из шести заключается в том, что хлеб присутствует почти везде. Чтобы принять это во внимание, мы определим *уверенность* как количество сделок, в которых множество элементов делится на общее количество раз, когда присутствует элемент.

Определение: уверенность

$$c(X \rightarrow Y) = \frac{|T(X\ AND\ Y)|}{T(X)},$$

где *T(X)* = это множество транзакций, где присутствует *X*.

Давайте посчитаем, что рейтингом доверия является то, что молоко будет в корзине, когда хлеб также купили. Это можно записать так:

$$c(\textit{хлеб} \rightarrow \textit{молоко}) = \frac{|T(\textit{хлеб AND молоко})|}{T(\textit{хлеб})}.$$

Далее вам нужно найти все сделки, содержащие и хлеб, и молоко, а затем только хлеб.

T (*хлеб AND молоко*) = {*молоко*,*хлеб*,*морковь*},{*хлеб*,*молоко*},{*молоко*,*финики*,*йогурт*,*хлеб*}

T (*хлеб*) = {*молоко*,*хлеб*,*морковь*},{*хлеб*,*морковь*},{*хлеб*,*молоко*},{*молоко*,*финики*,*йогурт*,*хлеб*}

Вставив это в уравнение, вы получите:

$$c(\textit{хлеб} \rightarrow \textit{молоко}) = \frac{|T(\textit{хлеб AND молоко})|}{T(\textit{хлеб})} = \frac{3}{5} = 0{,}6.$$

Согласно этим расчетам, вы можете быть на 60 % уверены, что найдете в корзине молоко, если в ней есть хлеб. Звучит хорошо, верно? Но притормозите. Если мы сделаем то же самое с финиками и морковью, тот же расчет даст вам

$$c(\textit{финики} \rightarrow \textit{морковь}) = 0{,}5.$$

Хотя я понимаю, что хлеб и молоко часто идут рука об руку, я не столь уверен, что люди, которые едят финики, также в половине случаев покупают морковь. Здесь недостаточно транзакций с финиками, чтобы поддержать это заявление. Это приводит ко второму определению, которое позволит понять, существует ли правило об ассоциации между двумя элементами.

Определение: поддержка

$$S(X \rightarrow Y) = \frac{|T(X\ AND\ Y)|}{T(X)},$$

где *T(X)* – это совокупность операций, которые содержат *X*, а *T()* означает все транзакции.

Наши два примера дают нам следующую поддержку:

$$S(\textit{хлеб} \rightarrow \textit{молоко}) = \frac{3}{6},$$

$$S(\textit{финики} \rightarrow \textit{морковь}) = \frac{1}{6}.$$

Иными словами, данных, которые поддерживают правила ассоциации *хлеб* → *молоко* гораздо больше, чем для правила *финики* → *морковь*. Но если мы обратимся к реальной жизни, то большинство магазинов (по крайней мере, те, которые выживают) продают больше шести продуктов, и каждая транзакция может быть гораздо больше.

Когда я начал писать эту главу, я спросил своих друзей на Facebook, что они купили в последний раз в супермаркете, надеясь получить хороший пример данных. Но обратная связь была слишком запутанной и не сработала. На рис. 5.13 также показано, что ассоциативные правила могут быстро сложнее в расчетах.

Для поиска ассоциативных правил вам в первую очередь необходимо найти частотные множества. На рис. 5.14 показаны возможные множества, которые могут появиться, если у вас есть множества из четырех элементов {молоко, масло, финики, хлеб}.

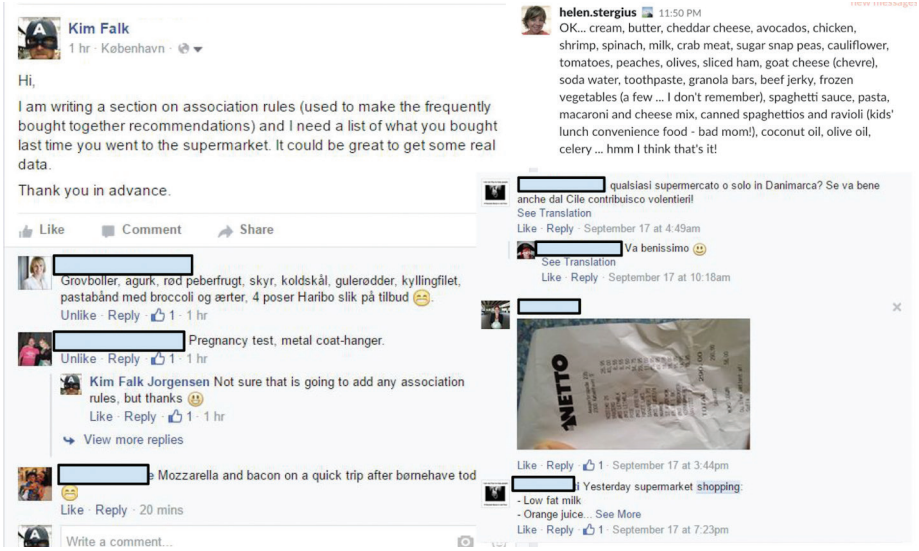


Рис. 5.13. Полезные друзья на Facebook оперативно откликнулись на мою просьбу

Опять же, это простой пример, но, чтобы объяснить, как как все это реализуется, нам нужна схема (рис. 5.14).

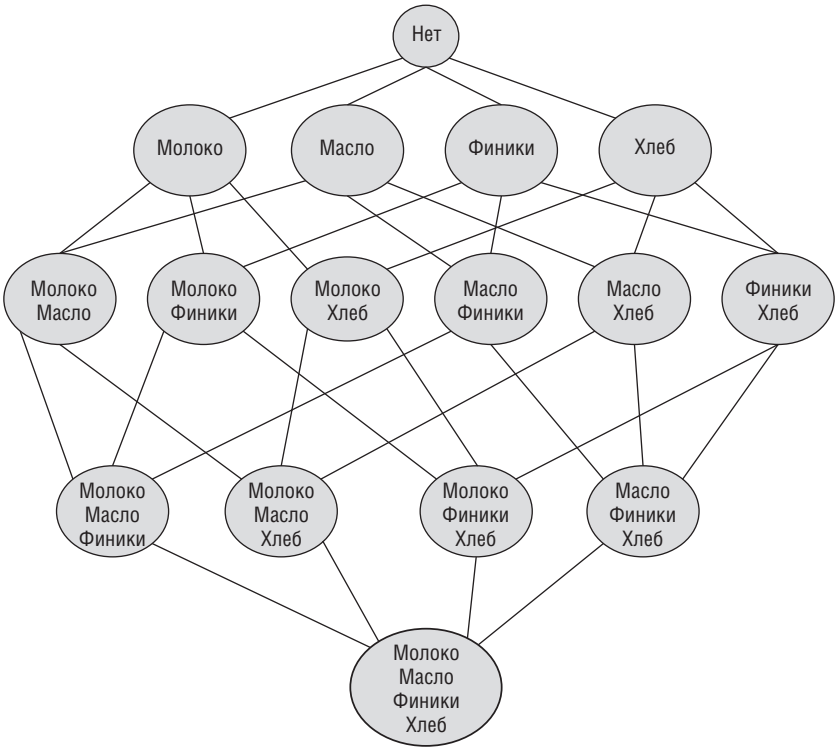


Рис. 5.14. Множества с четырьмя элементами

На рис. 5.14 показаны все возможные комбинации четырех элементов. Если начать из нижней части, где все элементы присутствуют в одной ячейке, то зададим вопрос: что должно произойти, чтобы это множество существовало? Все элементы должны часто присутствовать в одной транзакции, но тогда получается, каждый из элементов тоже часто присутствует в этом множестве.

Поднимемся на шаг вверх и выберем узел на пути к левой части {молоко, масло, финики}, как показано на рис. 5.15. Чтобы он стал частотным множеством, молоко, сливочное масло и финики должны часто возникать вместе. Это часто помогает выполнить быструю реализацию.

Рис. 5.15. Элемент сетки с малым присутствием масла

Начнем с изучения множеств из одного элемента, чтобы увидеть, как часто они появляются. Если вы обнаружили, что масло почти всегда отсутствует в множестве, то вы знаете, что множеств с маслом у нас не будет. Это означает, что можно вычеркивать все правила, которые содержат масло. Черные узлы на рис. 5.16 показывают, какие элементы можно удалить из списка, так как масло покупают редко. Имея это в виду, посмотрим на реализацию в следующем разделе.

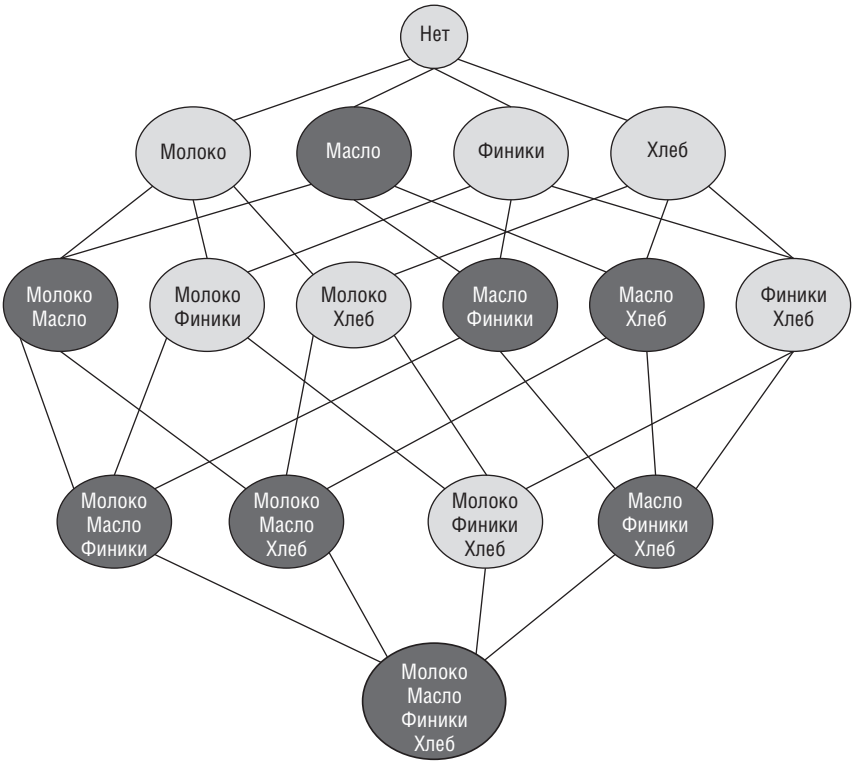


Рис. 5.16. Черные узлы на диаграмме показывают, какие множества никогда не будут производить какие-либо правила ассоциации. Поскольку масло встречается редко, ни один из узлов, содержащих масло, не может быть частым

5.4.3. Реализация ассоциативных правил

Процедура, описанная в предыдущем разделе, выглядит примерно так:

1. Выбор на минимальной поддержке и минимальном уровне уверенности.
2. Получение всех транзакций.
3. Создание списка множеств, по одному для каждого элемента, вычисление их поддержки (количество присутствий на количество сделок) и установка уверенности на уровень 1.
4. Создание списка множеств, содержащих более одного элемента, вычисление поддержки и уверенности так, чтобы каждая сделка вносила вклад в поддержку элемента.
5. Перебор множеств данных и удаление недостаточно «доверенных» множеств.

Превратим это в код на Python, но пока повременим с выбором минимальной поддержки и уровня доверия и выполним вычисления.

Получение всех сделок

На сайте MovieGEEKs нет понятия корзины, так что вместо этого мы будем считать покупки, которые произошли в одной сессии, транзакциями. Вы получите ваши транзакции из журнала, поэтому сначала нужно извлечь события покупок из базы данных, а затем выстроить транзакции на основе идентификатора сессии, как показано на рис. 5.17.

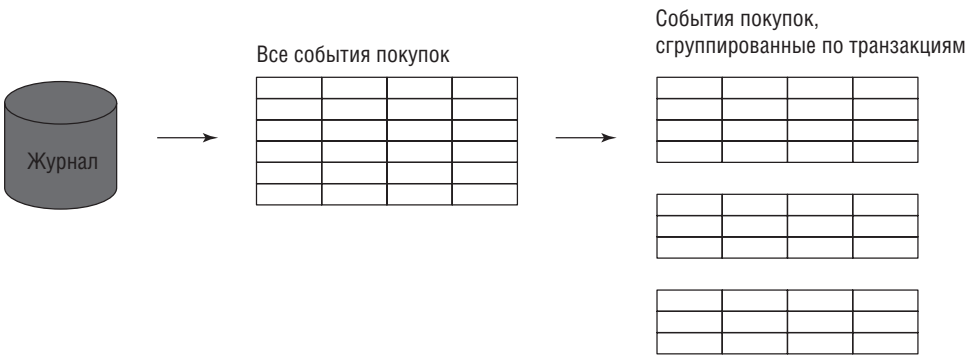


Рис. 5.17. Создание транзакций по данным журнала

Чтобы получить все транзакции из вашего журнала, выберите все записи, которые содержат события покупок, как показано в листинге 5.6. Это реализует метод `retrieve_buy_events`.

ЛИСТИНГ 5.6. Извлечение событий покупок из журнала

```
def retrieve_buy_events():

sql = """
SELECT *
FROM  Collector_log
WHERE event = 'buy'
ORDER BY session_id, content_id
"""
    cursor = data_helper.get_query_cursor(sql)
    data = data_helper.dictfetchall(cursor)

    return data
```

Используется з прос SQL для извлечения событий покупок

Теперь нужно сгруппировать все события покупок в сделки. Чтобы сделать это, нужно подать данные в метод `generate_transactions`, как показано в листинге 5.7. Этот метод работает с данными и собирает все транзакции в словарь, который содержит идентификатор транзакции в качестве ключа и несколько идентификаторов сессий для транзакций.

ЛИСТИНГ 5.7. Создание списка и добавление в словарь

```
def generate_transactions(data):
    transactions = dict()

    for trans_item in data:
        id = trans_item["session_id"]
        if id not in transactions:
            transactions[id] = []
        transactions[id].append(trans_item["content_id"])

    return transactions
```

Перебор всех строк данных

Извлекает идентификатор транзакции (идентификатор сессии в данном случае)

Если не сделано, создается список и добавляется в словарь

Добавляется контент к транзакции

Извлечение всех множеств элементов размера 1 и вычисление поддержки

Теперь можно вычислить частотные множества. Способ, показанный в листинге 5.8, позволяет легко представить, что происходит.

ЛИСТИНГ 5.8. Расчет поддержки и уверенности для частотных множеств

```
def calculate_support_confidence(transactions, min_sup=0.01):
    N = len(transactions)

    one_itemsets = calculate_itemsets_one(transactions, min_sup)
    two_itemsets =
     calculate_itemsets_two(transactions,
                            one_itemsets, min_sup)
    rules = calculate_association_rules(one_itemsets,
                                        two_itemsets, N)

    return sorted(rules)
```

N – число транзакций

Расчет всех множеств размера 1

Расчет правил ассоциации

Расчет всех множеств размера 2

Сортировка правил

Метод создает два словаря: один для частотных множеств с одним элементом и один для множеств с двумя элементами. После деклараций идет два вызова методов, которые заполняют словари. Давайте сперва посмотрим на `calculate_itemsets_one`.

ЛИСТИНГ 5.9. Расчет списка множеств элементов с одним элементом

```
    def calculate_itemsets_one(transactions, min_sup=0.01):
        N = len(transactions)

        temp = defaultdict(int)
        one_itemsets = dict()

        for key, items in transactions.items():
```

N – число транзакций

`defaultdict` инициализирует новые элементы со значениями по умолчанию входного типа

Проход через все транзакции

```
    for item in items:
        inx = frozenset({item})
        temp[inx] += 1

# удаляем все неподдерживаемые элементы.
for key, itemset in temp.items():
    if itemset > min_sup * N:
        one_itemsets[key] = itemset

return one_itemsets
```

Просмотр к ждого элемент

Проход по всем элемент м

Выбор элементов со зн чением поддержки выше требуемого

Поскольку используется **defaultdict**, об иници лиз ции беспокоиться не стоит

FrozenSets - это особый тип множеств. Они могут использов ться в к честве ключей для слов рей

Словарь defaultdict

В предыдущем листинге импортируется `defaultdict`. Это словарь, где каждый новый элемент инициализируется значением по умолчанию для заявленного типа. Это делает код немного более читаемым. Метод проходит через все транзакции, и для каждой транзакции он увеличивает количество для каждого найденного элемента в транзакциях.

Когда все будет готово, множества с одним элементом (как в листинге 5.9) подаются в этот метод, который вычисляет множества элементов с уверенностью и с поддержкой, превышающей некоторое минимальное значение. В листинге 5.10 показан расчет для множеств с двумя элементами.

ЛИСТИНГ 5.10. Создание списка множеств с двумя элементами

```
def calculate_itemsets_two(transactions, one_itemsets, min_sup=0.01):
    two_itemsets = defaultdict(int)

    for key, items in transactions.items():
        items = list(set(items))

    if (len(items) > 2):
        for perm in combinations(items, 2):
            if has_support(perm, one_itemsets):
                two_itemsets[frozenset(perm)] += 1
    elif len(items) == 2:
        if has_support(items, one_itemsets):
```

Проход по всем тр нз кциям

Уд ление дублей

Проверк , есть ли у множеств поддержк

Проверк тр нз кций с двумя элемент ми

Просмотр всех сочет - ний двух элементов для формиров ния списк

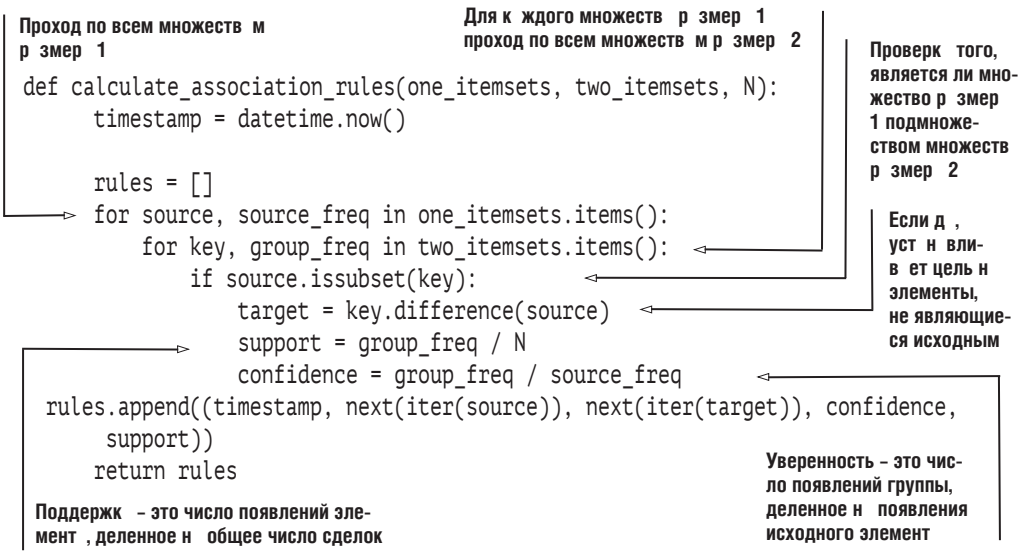
Доб вление множеств к списку элементов

[1] Для дополнительной информации см. **mng.bz/o2h5**.

```
                    two_itemsets[frozenset(items)] += 1
    return two_itemsets
```

Получаемый словарь просматривается еще раз, и элементы со значениями поддержки выше минимального добавляются к выходному словарю, который в конечном итоге возвращается. Теперь вы готовы к вычислению ассоциативных правил (листинг 5.11).

ЛИСТИНГ 5.11. Расчет ассоциативных правил



Стоит отметить, что, возможно, стоит найти в ассоциативных правилах множества с более чем один элемент в левой части правила, если вы хотите использовать его, чтобы рекомендовать что-то, основываясь на корзине покупок. Но такой расчет с использованием только одного исходного элемента в данном случае имеет смысл, потому что вы будете использовать его, чтобы показать рекомендации, связанные с одним элементом.

5.4.4. Сохранение ассоциативных правил в базе данных

Теперь, когда вы умеете вычислять ассоциативные правила, стоит подумать о том, нужно ли вычислять их каждый раз, когда клиент смотрит на продукт. Вы можете вычислять правила в автономном режиме, а затем как-то быстро извлекать их. Но ассоциативные правила нужно обновлять, и их обновление не должно нарушать работу.

Можете ли вы сохранять ассоциативные правила в одной таблице? Тут есть свои проблемы. Давайте сделаем шаг назад, чтобы вы успели за мыслью. Ваши пользователи щелкают по подробной информации о фильмах, что создает за-

прос в таблицу правил. Это также происходит в момент, когда система добавляет новые правила. Для того чтобы избежать проблем при сохранении новых правил, вам нужен маркер, который показывает, какие правила нужно извлекать.

Способ обойти это – ввести таблицу версий, как показано на рис. 5.18. Таблица версий гарантирует, что система не будет смешивать различные версии правил. В таблице версий содержится строка для каждой полной версии правил. Это означает, что вы не можете запросить правила ассоциации напрямую, но можете присоединиться к ним через таблицу, как показано в листинге 5.12.

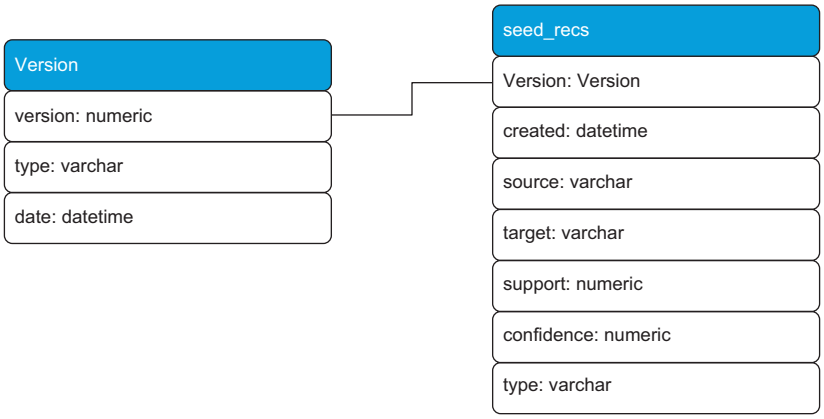


Рис. 5.18. Информационная модель правил ассоциации

ЛИСТИНГ 5.12. Код SQL для извлечения правил ассоциации в соответствии с версией

```
WITH currentversion as
 (SELECT version
     FROM version
     WHERE type = 'association_rules'
     ORDER BY version desc
   LIMIT 1)
SELECT *
FROM seeded_recs recs
WHERE source = '<the source id>'
AND recs.version = currentversion
```

Но не сильно радуйтесь таблицам версий, так как на сайте MovieGEEKs они не реализованы.

5.4.5. Запуск калькулятора ассоциаций

Для запуска калькулятора ассоциаций вам сначала нужно создать записи журнала, о которых мы говорили в главе 4 (запустив файл *populate_logs.py*). Затем можете запустить код, приведенный в следующем листинге.

ЛИСТИНГ 5.13. Расчет ассоциативных правил

```
python -m builder.association_rules_calculator
```

Этот код создает ассоциативные правила и сохраняет их в базе данных. Здесь вам нужно понимать, как работают и реализуются ассоциативные правила.

Давайте быстро пробежимся по сайту MovieGEEKs, чтобы увидеть правила в действии. Для получения рекомендаций с использованием ассоциативных правил нужно вызвать метод, приведенный в листинге 5.14. Это код Django (и возможно, он не слишком интересен для вас). Но нам все равно нужно иметь возможность видеть, что происходит.

ЛИСТИНГ 5.14. Составление правил выборочных рекомендаций

```
def get_association_rules_for(request, content_id, take=6):
    data = SeededRecs.objects.filter(source=content_id) \
                .order_by('-confidence') \
                .values('target', 'confidence', 'support')[:take]
    return JsonResponse(dict(data=list(data)), safe=False)
```

Извлекает объекты из таблицы SeededRecs по идентификатору контента и сортирует их по уровню уверенности

Обернув в json и возвращает

На лендинг-странице, показанной на рис. 5.19, справа теперь выводится таблица топ-10. (Рекомендации на этой странице также взяты из правил ассоциации, но об этом поговорим в главе 6.) Если щелкнуть по записи в таблице (в теории вы можете нажать на любой фильм, но рекомендации построены на небольшом наборе данных, поэтому они будут появляться только в нескольких местах), вы увидите ассоциативные правила на странице **Подробностей**.

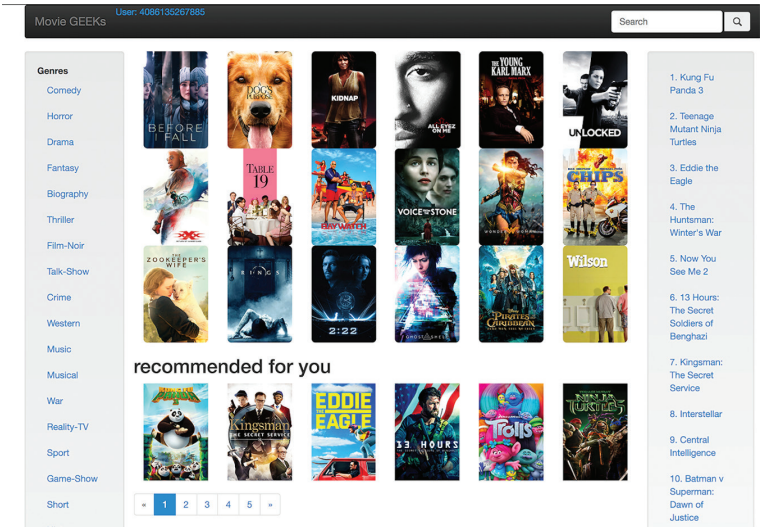


Рис. 5.19. Лендинг с топ-10 контента справа

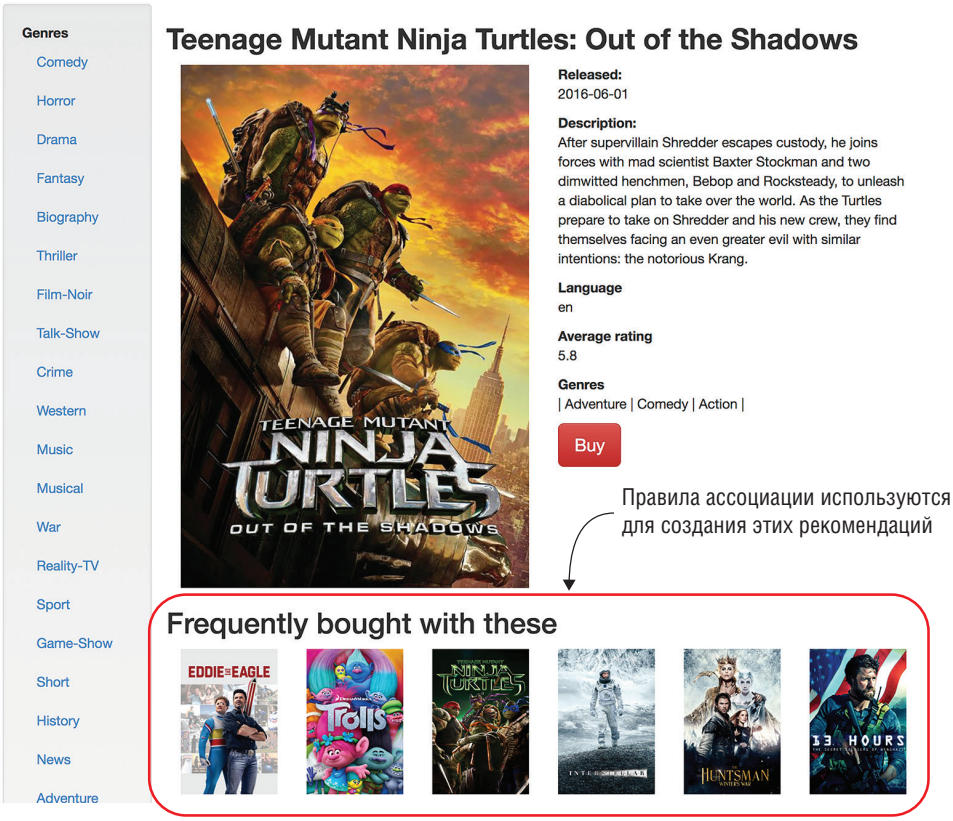


Рис. 5.20. Рекомендации на странице с подробной информацией

В моем случае, когда я выбираю «Черепашек ниндзя», я получаю рекомендации, показанные на рис. 5.20.

5.4.6. Использование различных событий для создания ассоциативных правил

Друг посоветовал мне почитать книгу «Online Consumer Behavior» Анджелины Клоуз (Routledge, 2012), чтобы получить более глубокое представление о том, как пользователи ведут себя в интернете, а затем добавить это в главу 3 (попробуйте угадать, сделал ли я это). Читая эту книгу, я наткнулся на новый тип рекомендации на Amazon, которого не видел раньше (рис. 5.21). Я думаю, что эта рекомендация оказывает книге медвежью услугу, потому что возникает ощущение, что люди искали книгу, а купили что-то другое.

Однако здесь показан способ усилить ваши ассоциативные правила. Даже если не так много людей купили эту книгу, вы по-прежнему можете создавать ассоциативные правила путем определения всех сессий, где клиент просматривал книгу, а затем взглянуть на то, что клиент в конце концов купил.

То есть отправной точкой здесь будут не только покупки товара, а еще и его просмотры.

Рис. 5.21. Рекомендатор Amazon: что клиенты купили после просмотра товара.

Затем, когда вы рассчитаете все частотные множества, вы сможете вычислить правила и использовать их в качестве рекомендаций.

Все это так интересно, что нам даже не нужно подводить итог главы, и вы, вероятно хотите скорее перейти к следующей главе и приступить к созданию рекомендаций. Но резюме – хороший способ освежить прочитанное.

Резюме

- Графики – это круто, и их легко добавить. Вы можете построить данные разными способами, а не только путем подсчета того, что покупалось наиболее часто.
- Лучше не отображать неупорядоченный контент. Контент должен быть отсортирован для удобства пользователей по вашему разумению. Для фильмов и книг это может быть сортировка по дате выпуска, а для купонов – по размеру скидки.
- Ассоциативные правила позволяют анализировать, что покупается вместе, и используются для формирования FBT-рекомендаций. Полезность правил рассчитывается с учетом значений поддержки и уверенности.
- Рекомендации стоит сохранять в базе данных. Это позволит добиться от системы более быстрой реакции. С другой стороны, их вычисление отнимает много времени, и они занимают место.
- Добавление номера версии к рекомендациям в базе данных позволяет одновременно иметь несколько версий базы, а это значит, что вы можете иметь версию для реальной работы, а другую на будущее. Но что еще более важно, если что-то случится с рекомендациями, которые вы в данный момент используете, вы сможете вернуться к более старой версии.

Глава 6

«Холодные» пользователи и контент

Раскройте объятия и улыбнитесь – пришло время поприветствовать ваших новых клиентов! В этой главе мы:

- исследуем проблему холодного старта, которая связана с новыми клиентами;
- узнаем, как сегментировать пользователей, чтобы создавать полуперсонализированные рекомендации;
- рассмотрим сайт Redbubble.com как случай проблемы холодного старта, но уже с вновь приобретенными знаниями;
- рассмотрим реализацию простого персонализированного рекомендатора с использованием ассоциативных правил.

Наденьте шапку и перчатки – у нас холодный старт! В предыдущей главе мы говорили о том, как получить данные, и, к счастью, большинство сайтов сперва собирают данные, и лишь потом начинают приключения в мире рекомендательных систем. Но даже наличие данных не решит проблему ввода новых продуктов или пользователей.

6.1. Что такое холодный старт?

Неудивительно, что, если у вас нет знаний о ваших пользователях, вы не можете персонализировать их. Отсутствие персонализации – это огромная проблема, потому что нам хочется, чтобы новые посетители чувствовали себя желанными, и именно так они становятся лояльными постоянными клиентами. Постоянные потребители идеальны, и их нужно поддерживать счастливыми, но добавление новых – совсем другое.

Эта проблема настолько велика, что у нее есть название – *холодный старт*. Термин касается не только генерации рекомендаций для новых пользователей, но и введения новых элементов в каталог. Новые элементы не будут ото-

бражаться в неперсонализированных рекомендациях, потому что им не хватает популярности, чтобы войти в статистику продаж, и они не будут появляться в персонализированных рекомендациях, потому что система не знает, как эти элементы связаны с другими.

В этой же связи вводятся *серые овцы*. Это пользователи, у которых столь изощренные индивидуальные вкусы, что даже при наличии данных не найдется других пользователей, которые купили бы что-то подобное.

Персонализированные рекомендации основаны на информации, которая связывает контент с пользователями. На рис. 6.1 показаны наиболее распространенные связи, используемые при расчете рекомендаций. В следующих главах мы поговорим о них подробнее, а здесь я лишь хочу сказать, что проблема холодного старта требует от нас выяснить, что делать, если у вас нет или мало этих типов связей.

На рисунке показано, что если пользователь высоко оценил фильм #1, то при наличии одной из этих связей вы можете порекомендовать фильм #2. Если таких связей нет, придется как-то придумывать, что рекомендовать.

К счастью, это происходит только тогда, когда у вас появляются новые пользователи, которые еще не связаны с контентом (ничего не оценили и не купили). И, как вы увидите, пользователи с уникальными вкусами создают примерно те же проблемы. Но я забегаю вперед.

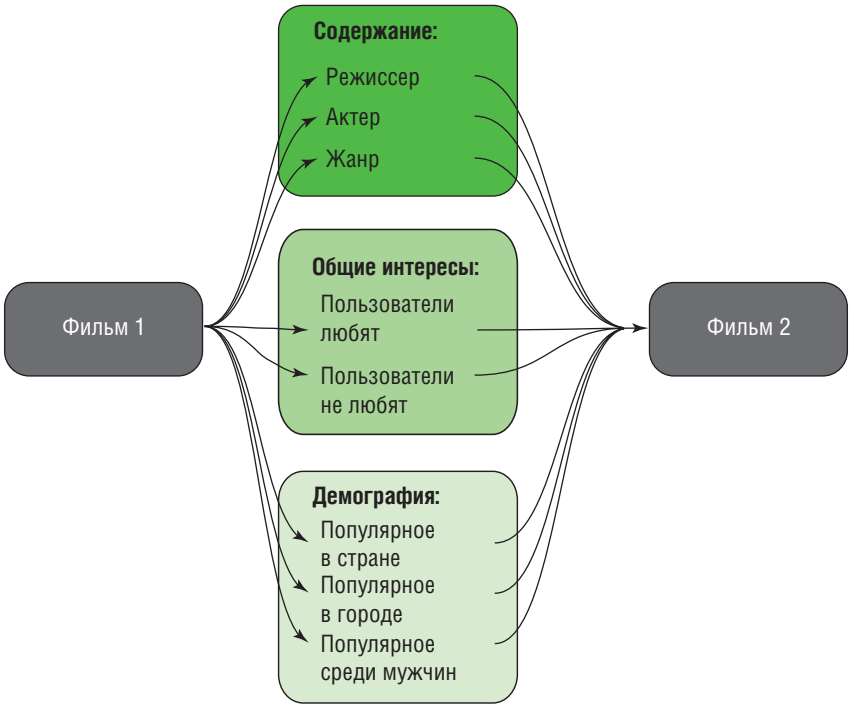


Рис. 6.1. Общие связи, используемые при расчете рекомендаций

Давайте притормозим и начнем с простого – холодных товаров.

6.1.1. Холодные товары

Каталогу товаров необязательно быть большим, чтобы новый контент мог затеряться как иголка в стоге сена. Поэтому крайне важно прикладывать дополнительные усилия, прежде чем привнести что-то новое. В большинстве случаев добавление нового контента должно сопровождаться ручным процессом продвижения элемента, например отправкой электронной почты пользователям с похожими интересами.

Самый простой способ заставить посетителя заметить новый контент – это добавить место на странице, где он будет выводиться. Большинство людей любят что-то новенькое. У Netflix есть отдельное место для демонстрации вновь прибывшего контента, как показано на рис. 6.2.

Рис. 6.2. Netflix выводит новые поступления

Другой способ продвинуть новый контент – это заставить его выглядеть популярным и выводить его в ваших рекомендациях. Тогда, если он не будет потребляться, его можно спускать вниз.

6.1.2. Холодный посетитель

Новый посетитель, о котором вы ничего не знаете, – это тоже холодный старт. Когда можно начать давать пользователю персональные рекомендации? Многие научные работы говорят, что нельзя рассчитать рекомендации до того, как пользователь оценит хотя бы 20–50 элементов. В нормальной ситуации клиент ожидает, что сайт начнет рекомендовать ему что-то задолго до этого.

Некоторые сайты с фильмами просят вас оценить пять элементов, прежде чем начать работу. Но для большинства сайтов это не вариант. Однако, если пользователь ищет что-то, это будет отличный способ понять, чего человек хочет, и это можно использовать для рекомендаций.

На рис. 6.3 показан мой поиск по LinkedIn. И тут же, в верхнем углу, где обычно и бывает, – кнопка **Create Job Alert**. Такие элементы могут дать хорошие данные относительно того, что пользователь хочет, и этого может быть достаточно, чтобы классифицировать одну или две группы контента, которые могут иметь отношение к пользователю.

Когда у вас есть достаточно информации для отображения, рекомендация является компромиссом. Это вопрос принятия решения о том, следует ли ото-

бражать рекомендации высокого качества, если у вас много данных, или более низкого качества с меньшим количеством данных.

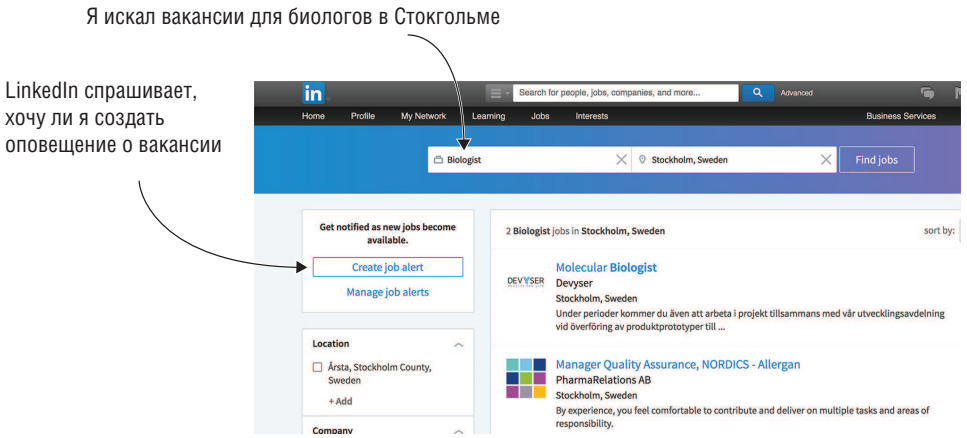


Рис. 6.3. LinkedIn выводит оповещение о вакансии, используя ваш поисковый запрос, что говорит о том, что поиск помог вам найти то, что нужно

Подумайте: каково наименьшее количество данных, которых будет достаточно для определения вкусов? В некоторых случаях может быть достаточно пяти оценок, как на сайтах фильмов, но в других местах это число может быть различным.

При поступлении нового пользователя вы не знаете о нем ничего[1]. Отсутствие данных означает отсутствие персонализации. Это легко, но, вместо того чтобы остановиться и успокоиться, давайте посмотрим, что вам нужно знать, прежде чем делать что-то.

Нужно иметь в виду, что, если у вас мало данных о пользователе, вы не сможете получить правильное представление о его предпочтениях. Например, представьте, что Сара заходит на MovieGEEKs и ее предпочтения в фильмах примерно такие: экшн: 20 %; драма: 20 % и комедии: 60 %, но оценок она еще не ставила. Сегодня она знает, чего хочет, и открывает жанр **Драма** и покупает фильм. Система знает, что Сара любит драмы, но не знает, насколько. Если рекомендатор предположит, что Сара смотрит только драмы, то он ошибется. Но, как уже упоминалось ранее, часто лучше слегка не угадать, чем вообще не рекомендовать ничего.

В главе 12 вы узнаете о гибридных рекомендаторах, а именно о смешанных гибридах. Когда используется такой рекомендатор, он возвращается к наиболее популярной рекомендации, если персональные рекомендации оказываются недоступны.

Например, Amazon будет в числе прочего показывать вам, что люди смотрят прямо сейчас. Реализовать это можно довольно быстро. Нужно просмотреть

[1] Хотя знание IP-адреса все же несет некоторую информацию. Подробнее об этом – в разделе 6.3.

журнал и найти контент, который просматривался в последнюю минуту, час или день. Но есть и другие вещи, которые вы можете сделать.

Даже зная что-то своих клиентах, вы все равно можете столкнуться с проблемой холодного старта. Мы называем это серыми овцами.

6.1.3. Серые овцы

Серая овца – это пользователь, который имеет настолько особенный вкус, что даже если о нем есть данные, то, скорее всего, нет или мало других потребителей, которые имели бы тот же вкус.

Это тоже проблема холодного старта, потому что для таких пользователей тоже очень сложно создать рекомендации. Некоторые решения подходят и для холодных посетителей, и для серых овец, поэтому поговорим об этом.

6.1.4. Посмотрим на примеры из реальной жизни

Возьмите фильм *X*, который никто не удосужился посмотреть. И тут приходит Сьюзен, которая, возможно, встретила кого-то, кто в течение двух минут играл в массовке фильма *X*, и именно эти две минуты она хочет посмотреть. Она болтала с этим человеком в интернете, хочет получить фильм и заходит на сайт фильма. В этом случае Сьюзен, вероятно, не нужны другие подобные фильмы, и рекомендовать ей что-то было бы ошибкой.

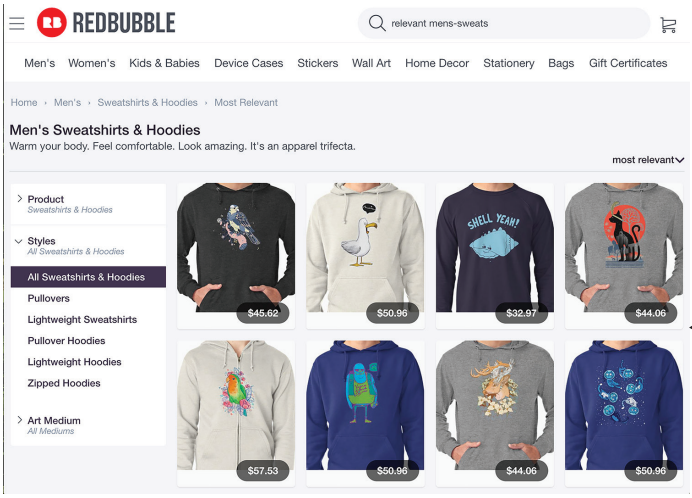
Как правило, люди покупают то, что им нравится. На самом деле Сьюзен создает больше проблем по двум причинам: система будет тратить силы на поиск подобия, но не найдет его, и, если Сьюзен потом купит что-то более популярное, внезапно образуется связь между популярным фильмом и тем, что было когда-то куплено одним пользователем.

Это может в конечном итоге заставить ваш рекомендатор начать рекомендовать фильм *X* тем, кому нравится тот самый более популярный фильм. Только после того, как пользователь сгенерирует больше данных, вы поймете, что фильм *X* попал сюда случайно и его надо было игнорировать. Это не проблема, когда мы говорим о правилах ассоциации, но, когда мы говорим о совместной фильтрации, это может стать проблемой.

Пример серых овец и холодных продуктов

Серые овцы – это такие странные пользователи, которым вы ничего не можете рекомендовать. Но рассмотрим сайт Redbubble (**www.redbubble.com**) (рис. 6.4). Redbubble – это место, где художники могут демонстрировать и продавать искусство во многих формах – от граффити до футболок. Имея каталог контента, содержащий миллионы произведений искусства, и клиентов со всего мира, Redbubble может похвастаться большим количеством таких «овец».

Искусство трудно классифицировать, поэтому трудно сказать, что понравится пользователю. Добавим сюда факт, что большинство элементов продается всего несколько раз, в результате чего становится трудно рекомендовать вещи даже старым клиентам.



Redbubble – это маркетплейс, куда художники загружают картинки, а Redbubble печатает их на различных материалах, таких как толстовки, наклейки или канцелярские принадлежности

Рис. 6.4. Главная страница Redbubble

Большинство продуктов Redbubble остается холодным, потому что мало человек покупает каждый продукт. Соответственно, связей между элементами нет. Клиенты, которые покупают на Redbubble, легко могут быть серыми овцами, а это означает, что они купили вещи, которые не были приобретены другими людьми.

Redbubble имеет ту же проблему, как новостные сайты, такие как Google-новости, видеохостинги вроде YouTube или журналы, как Issuu (приложение YouTube для журналов). Художники, кинематографисты или издатели журналов добавляют контент без описаний (например, жанр, теги и т. д.). Когда они добавляют информацию в виде тегов, люди интерпретируют значение тега по-разному. Но Redbubble особенно интересен тем, что поставить тег на элементы искусства сложно. Изучите Redbubble и посмотрите, сможете ли вы придумать способы для автоматического добавления тегов.

6.1.5. Что вы можете сделать с холодным стартом?

В следующих разделах мы рассмотрим различные способы, позволяющие сделать холодный запуск менее проблематичным. Многие из решений проблемы холодного старта (пользователи, продукты и серые овцы) являются одним из алгоритмов, которые вы будете изучать в следующих главах, так что это мы рассмотрим. Например, холодный элемент лучше всего обрабатывается с помощью фильтрации на основе контента, которую мы рассмотрим в главе 10. А пока я буду откладывать любые попытки решить эту проблему.

Если не решать проблему холодного старта алгоритмами, можно предложить людям авторизацию с помощью социальных сетей, например Facebook, а затем извлечь информацию о пользователе из его профиля.

Не факт, что из профилей вы автоматически получите данные, нужные именно для вашей задачи, но для начала сойдет.

6.2. Отслеживание посетителей

К сожалению, большинство пользователей неуловимы. Они, как правило, заходят на сайты без авторизации, с разных устройств из разных мест, так что иногда сайту сложно узнавать постоянных клиентов. Это печально, потому что сайту нужно отличать новых пользователей от старых. Вы должны отслеживать пользователей – и новых, и старых, – чтобы понять их поведение.

6.2.1. Анонимные пользователи

Как только новый пользователь заходит на ваш сайт, хорошо бы сразу объяснить ему преимущества регистрации и авторизации (через соцсети или форму на сайте). Вы предпочитаете зарегистрированных пользователей, но все равно нужно приберечь идентификаторы анонимных сессий, чтобы распознать пользователей, если они вернутся. Распознавание пользователей означает, что вы можете накапливать информацию о человеке, чтобы система рассчитывала рекомендации еще до того, как вы узнали пользователя.

В старые времена, когда у людей был один стационарный компьютер без каких-либо других устройств, было достаточно загрузить файлы куки в браузер пользователя. Это по-прежнему стоит делать, но помните, что, как только пользователь поменяет браузер или устройства, вы его потеряете. Эта проблема настолько важна, что компании рекламируют свои решения для нее[1].

Django разрешает анонимные сессии: это сессии, где вы можете загрузить куки, даже если пользователь не добавил информации в систему. Структура сессии позволяет хранить и получать произвольные данные для каждого посетителя сайта. Он хранит данные на стороне сервера и абстрагирует отправку и получение куки.

Файлы куки содержат идентификатор сессии, который можно посмотреть в базе данных, где хранятся ваши данные. Задайте идентификатор пользователя и сохраните его. Как только получены куки, предполагая, что только один человек заходит с этого устройства, система может идентифицировать пользователя. Сохранение сессий на сервере требует осторожности, потому что, если у вас 40 млн пользователей, хранение и извлечение пользовательских данных становится проблемой. Реализовать отслеживание пользователей трудно, даже если это старые пользователи, но, если эта проблема решена, есть и другие подходы к холодному старту.

6.3. Решение проблемы холодного старта алгоритмами

Холодный старт до сих пор считается проблемой, потому что никто не придумал идеального решения для нее. Как бы ни вопили журналы, машинное обучение не творит чудес, оно всего лишь делает выводы из данных. Если нет

[1] На примере Adform: **blog.adform.com/products/cross-device-audience-management**.

данных с сигналом, то не будет и разумного ответа. Решение – поиск информации даже в разреженных данных.

Вы хотите использовать информацию, которая у вас уже есть, и связывать ее с новыми пользователями.

В следующих разделах вы будете использовать ассоциативные правила из предыдущей главы, а также посмотрите на создание сегментов для существующих пользователей, затем рассмотрите, как быстро разместить ваших пользователей в сегмент. Наконец, мы должны немного поговорить о том, как вы можете спросить у пользователя, что ему нравится.

6.3.1. Использование ассоциативных правил для создания рекомендаций для холодных пользователей

Ассоциативные правила создаются путем просмотра покупок создания правил, которые говорят рекомендатору, что если человек положил в корзину хлеб, то можно рекомендовать ему масло. В предыдущей главе эти правила были использованы для выработки рекомендаций по элементам. Но что, если мы вернемся назад и используем эти правила в сочетании с данными о новом пользователе (которых мало), а именно данными о его просмотрах. Затем вы используете их в качестве сидов, чтобы найти соответствующие правила, и из этих правил можно произвести рекомендации. На рис. 6.5 показано, как это может быть реализовано.

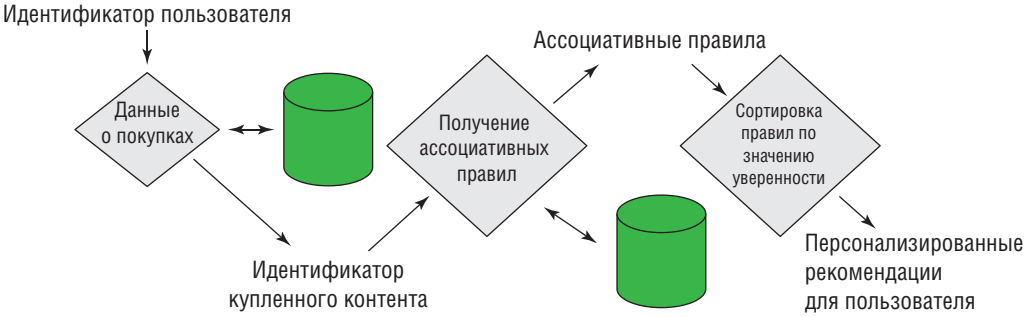


Рис. 6.5. Создание персонализированных рекомендаций с ассоциативными правилами

Система может запуститься, когда пользователь просмотрит первый элемент, что может произойти в первой сессии. Когда количество посещений увеличивается, данные могут стать все более и более ограничены, как показано в табл. 6.1.

Таблица 6.1. Какие события использовать

Количество посещений	События
0–2	Просмотр элементов
2–4	Подробная информация
4+	Покупка

Вы также могли бы определить их иначе, извлекая только ассоциативные правила, основанные на более ценной информации. Что-то вроде:

- получить все ассоциативные правила на основе покупок пользователя;
- получить все ассоциативные правила на основе покупок и просмотров подробностей пользователя;
- получить все ассоциативные правила, основанные на данных всех пользователей.

Средневзвешенное значение купленных товаров

Когда пользователь покупает свой первый элемент, вы сможете использовать ассоциативные правила для предоставления рекомендаций на основе этого элемента. И по мере покупок вы сможете брать средневзвешенные значения ассоциативных правил. Давайте предположим, что пользователь купил хлеб и масло, а система уже имеет следующие правила:

```
хлеб => мармелад[1]
хлеб => масло
хлеб => сыр горгонзола
масло => мармелад
масло => мука
яйцо => бекон
```

Затем, так как хлеб и масло были куплены, был бы следующий выбор:

```
хлеб => мармелад (и масло => мармелад)
хлеб => сыр горгонзола
масло => мука
```

Масло также было куплено, так что вы должны добавить правило `хлеб => масло`, но, так как масло уже куплено, это не добавит смысла. Если два правила указывают на одну цель, можно вычислить новое значение уверенности с использованием взвешенного среднего. Можно отсортировать их и взять лучшие, удалить дубликаты из списка и вернуть рекомендации. Если пользователь уже зарегистрирован, но для вас еще новый, так как у вас мало данных, вы можете также добавить веса к каждому из правил, основываясь на том, когда пользователь купил товар. Таким образом, более поздние покупки весят больше.

Чтобы рекомендации фильмов выглядели лучше, можно было бы добавить бизнес-правила. Это означает объединение знаний в предметной области в систему, которую мы рассмотрим далее.

---

[1] Правило `хлеб => мармелад` означает, что в данных есть закономерность, указывающая, что хлеб и мармелад часто покупаются вместе.

6.3.2. Использование знаний предметной области и бизнес-правил

Иногда рекомендатор не может делать все за вас. Он не может понять, что показывать, когда люди покупают мультфильмы и фильмы ужасов. В конце концов возникают ассоциации. Те, кто покупают «Бэмби» и «Техасскую резню бензопилой», могут заставить систему рекомендовать детям кровавые ужастики!

Одним из способов этого избежать является фильтрация контента, позволяющая ограничить рекомендации определенных типов контента в зависимости от того, что пользователь в настоящее время смотрит. Дата-сайентисты могли бы сказать, что вы таким образом вручную мешаете алгоритму работать, но иногда это необходимо.

Бизнес-правила могут быть определены положительно или отрицательно. Вы можете сказать: «Во время просмотра мультфильмов система может только рекомендовать мультфильмы и семейные фильмы». И наоборот – вы можете перечислить все вещи, которые не должны появляться в рекомендациях, «никогда не рекомендовать фильмы ужасов тем, кто смотрит мультики».

У вас есть несколько способов реализовать это. Обычно это делается путем вычисления списка из 100 рекомендаций (или 10), затем их фильтруют и берут топ-10 из оставшегося, как показано на рис. 6.6.

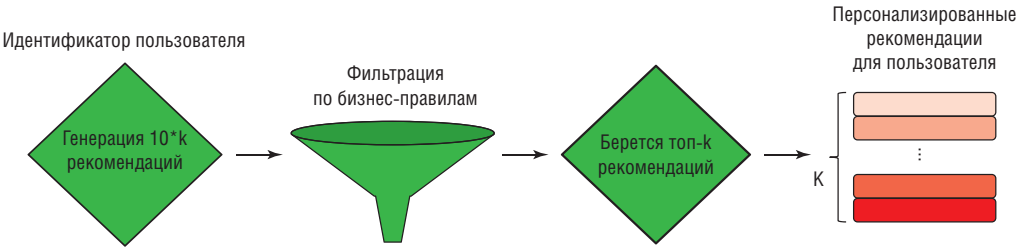


Рис. 6.6. Использование бизнес-правил для выдачи разумных рекомендаций

6.3.3. Использование сегментов

Холодный старт – это раздражающая проблема, но сделать можно мало что. Опять же, если вы знаете что-нибудь о пользователях, может ли это быть полезно для чего-то?

В Дании есть поговорка: «Похожие дети играют лучше». Я не уверен, что когда-либо понимал это выражение, потому что мне кажется, что противоположности тоже хорошо играют вместе. Но одно можно сказать точно: некоторым детям с похожими вкусами лучше смотреть телевизор вместе. Такая интерпретация заставит авторов пословицы перевернуться в могиле, но остановимся на ней.

Вам нужно сгруппировать людей с похожими вкусами, чтобы можно было выяснить, какой тип людей какой любит контент. Когда приходят новые посе-

тители определенного типа, вы можете рекомендовать контент, популярный у этого сегмента. Это известно как демографические рекомендации. Многие люди говорят, что это не работает, потому что люди слишком отличаются в своих демографических показаниях. Но зато мы приближаемся к полуперсонализированным рекомендациям. Получить эти данные часто невозможно без авторизации, но если она есть – это круто.

Очевидные сегменты

Перед тем как вывести большие пушки и начать делать *кластеризацию*, которая представляет собой обучение без учителя, давайте подумаем о том, какие сегменты наиболее очевидны. Если у вас есть посетитель, вы часто можете получить IP-адрес, а имея адрес, вы часто можете точно определить, где человек находится.

Если человек из Копенгагена и в вашем магазине есть вещи, которые не продаются в Дании, то вам лучше отфильтровать эти элементы и показывать только то, что продается в Копенгагене (столица Дании). Если вы продаете одежду, то будет, как правило, по крайней мере две группы: мужской и женский пол. Кроме того, вы можете также разделить пользователей на возрастные группы.

Давайте рассмотрим пример того, как вы могли бы воспользоваться данными пола. Если у вас имеется база данных пользователя, где есть пол человека, вы сможете фильтровать запрос при получении события покупок от пользователей и построить диаграмму, основанную на тех пользователях, вместо полного набора данных. То же самое справедливо и для мужчин. Хотелось бы отметить, что, вероятно, немало мужчин классифицируются системой как женщины после покупки платья в подарок и наоборот.

Имея магазин одежды, представьте, что у вас есть изображение, содержащее платья и мужские костюмы. Мужчине было бы легче, если бы выводили ему только мужскую одежду. Зная пол посетителя, вам нужно лишь отфильтровать контент. Но давайте двигаться дальше, и поговорим о контенте, который не имеет определенной гендерной специфики.

В качестве примера деления на возрастные группы, например, посмотрите на фильм «Звездные войны», который скучен для тех, кто на момент выхода был маловат или староват. Дети и поколение, которые смотрели их в кино, обожают эти фильмы. Если бы вы знали возраст ваших посетителей, вы могли бы рекомендовать нужные фильмы.

Неочевидные сегменты

Давайте уйдем от очевидных сегментов из предыдущего раздела и рассмотрим следующее: что, если бы у вас были группы типа «немка, которая любит по выходным боевики» или «американские подростки, покупающие фильмы ужасов во время учебы». Эти сегменты не так очевидны, и их может быть трудно обнаружить даже в данных, но это ценная информация для персонализации сайта для пользователей. Сегментация не должна включать в себя

демографические данные. Она может быть основана на любом виде данных пользователей.

Обычно сегменты создаются исследователями рынка на основе отраслевой практики и опыта. «Угадайка» – это хорошо, но не всегда практично. Таким образом, вместо того чтобы делать сегменты вручную, все больше и больше исследователей используют кластеризацию и выделяют не столь очевидные сегменты.

Кластерный анализ является менее субъективным способом поиска сегментов и может быть реализован с помощью *машинного обучения без учителя*. *Кластер* –это причудливое слово для группы с похожими признаками, поэтому мы постараемся найти конкретные типы пользователей, которые потребляют конкретный контент.

6.3.4. Использование категорий с целью обойти проблему серых овец и холодных продуктов

«Иногда нужно сделать шаг назад, чтобы пройти вперед». Эта пословица всегда раздражает, но иногда это необходимо. Идея лежит в основе следующего метода решения проблемы серых овец и холодных продуктов.

Если у вас есть ряд продуктов, которые купили или оценили всего несколько людей, трудно составить рекомендации. Но если вы делаете шаг назад и используете метаданные продукта, вы сможете найти подобные продукты. Это звучит странно, поэтому приведем пример.

Вернемся к Redbubble: художники стремятся нарисовать картинку, которая попадет под вкус определенной группы пользователей (по крайней мере, так я это предполагаю). Чтобы уменьшить разреженность данных, можно посмотреть на художников, а не на контент. Чтобы сделать это, вам нужно сгруппировать все художественные работы художника в один элемент.

Сальвадор Дали создал много произведений искусства, два из которых показаны на рис. 6.7. И если вы можете себе представить, что он не является всемирно известным и что тысячи будут покупать его картины, вы можете посмотреть на пользователей, которые купили картину слева, как на людей, которые также купили *X*. Или вы могли бы смотреть на пользователей, которые приобрели картину Сальвадора Дали и которые также купили картины художника *X*. И если решение о покупке основано на контенте, то можно рекомендовать наиболее популярные картины художника *X*.

В целом это можно представить так, как показано на рис. 6.8. Вы можете использовать эту логику для сайтов, таких как Redbubble, и для многих других сайтов. В случае с музыкой можно объединять песни по исполнителям, новости – по темам и т. д.

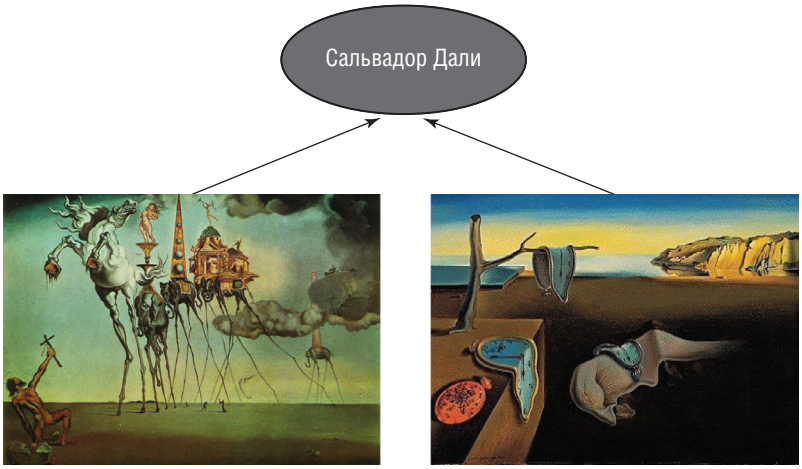


Рис. 6.7. Группирование картин, а не художников

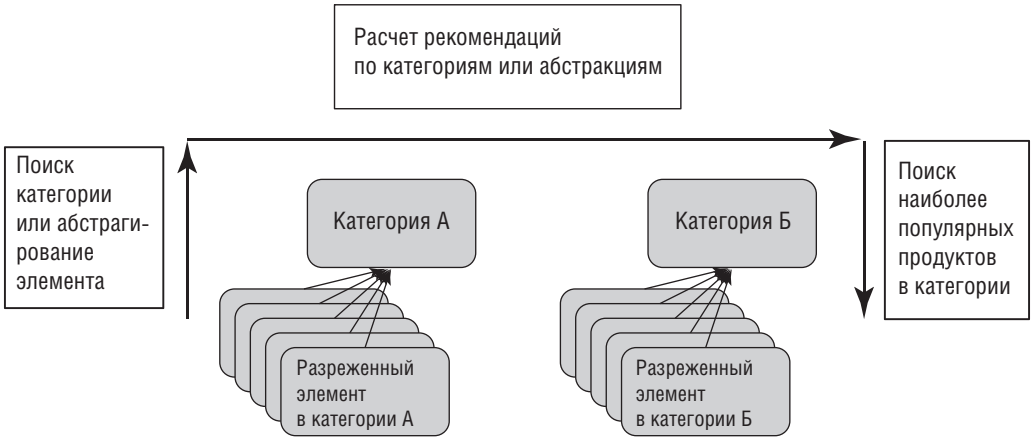


Рис. 6.8. Использование метаданных для решения проблемы разреженных данных

Помните, что абстракция или классификация не может иметь слишком общий характер, поскольку ассоциации между категориями теряют свою ценность. Примером такого объединения является `экшн => комедии`. Вы можете найти множество ассоциаций в данных, но для качества рекомендаций это может иметь маленькое значение.

Не столь очевидный пример может быть таким: я люблю фильмы Дж. Дж. Абрамса, поэтому я бы посмотрел любой его фильм (особенно после «Пробуждения силы»). Людям, которым он нравится, возможно, также нравится Ричард Маркуанд, снявший «Возвращение Джедая»[1]. Если вы реализуете это (но мы этого делать не будем), вы создадите ассоциативные правила на

[1] Ладно, «Звездные войны» от Марканда мне не очень понравились, но для примера оставим.

основе абстракции, а затем каждый раз, когда запрашиваете рекомендации, сможете использовать текущий элемент, который просматривает пользователь. Это зависит от набора данных.

Классификация может быть разной: произведения Толкиена или автомобили, проезжающие более 40 км за литр бензина. Чтобы реализовать это, вы можете изменить правила ассоциации, описанные в главе 5, и вместо сохранения идентификатора элемента сохранять автора каждой категории и составлять правила на основе метаданных.

6.4. Кто не спрашивает, тот не будет знать

Для начала можно спросить у пользователей, что они думают о выбранном списке элементов. Но для большинства сайтов электронной коммерции будет плохой идеей ограничивать доступ, пока посетитель не ответит на несколько вопросов. С другой стороны, вы можете сделать это необязательным, однако.

Amazon предлагает Вам возможность улучшить свои рекомендации. Для этого нужно авторизоваться и нажать кнопку, показанную на рис. 6.9. Это не слишком полезно в нашей проблеме, потому что Amazon собирает эти данные со старых, а не новых пользователей.

Рис. 6.9. Amazon предоставляет возможность улучшить свои рекомендации

Как создать страницу, на которой пользователь сможет рассказать вам о своих предпочтениях? Это не так просто, как звучит, потому что неясно, какой контент нужно показывать – наиболее популярный или наименее популярный? Если вы выберете второй вариант, скорее всего, вы получите серую овцу и не приблизитесь к пониманию пользователя. Возможно, стоит просто угадать рекомендации. Еще нужно подумать о том, что, если мы возьмем популярный элемент, этого пользователя можно будет сравнить с другими.

Чтобы решить эту проблему, вы должны обратиться к *активному обучению*, и это круто. Но к сожалению, это выходит за рамки данной книги. Вы можете почитать статью «Активное обучение» Рубена и соавторов[1]. Активное обуче-

[1] Rubens N., M Elahi, M. Sugiyama, D. Kaplan «Active Learning in Recommender Systems». Ricci F., L. Rokach, B. Shapira (eds) Recommender Systems Handbook (Springer, 2015).

ние для рекомендательных систем заключается в создании алгоритма, который дает пользователю контент для оценки, и рекомендатор в результате понимает информацию о предпочтениях человека.

6.4.1. Когда посетитель уже не новый

Время от времени стоит проверять пользователей на «новизну». Для такого рода рекомендаций можно использовать данные последней недели или последние 20 точек данных. Или вы можете добавить веса, чтобы новые элементы имели больший вес. Можно также сказать, что, когда пользователь купил пять элементов, вы будете использовать события покупки в качестве сидов.

Ассоциативные правила имеют смысл, когда вы смотрите на сырые данные, а не на неявные оценки. Можно, однако, начать с использования неявных оценок в качестве сидов, а затем взвешивать их на основе оценок.

6.5. Использование ассоциативных правил с целью ускорить показ рекомендаций

Как бы вы добавили ассоциативные правила на сайт MovieGEEKs? У вас уже есть готовый фреймворк для ассоциативных правил, так что вы должны принять то, к чему пользователь проявил интерес, и затем найти ассоциативные правила для каждого из этих элементов. После этого вы можете вернуть рекомендации, которые, по-вашему, подойдут лучше. Здесь не так много магии, но давайте посмотрим, что из этого получится.

Ассоциативные правила являются одним из способов реализовать это, и использовать можно любой метод поиска сходства. Например, вы могли бы использовать рекомендации на основе контента, но мы их еще не изучили, так что давайте начнем с ассоциативных правил. С точки зрения сайта MovieGEEKs вы будете использовать пространство под другими элементами на первой странице, как показано на рис. 6.10.

Вот список действий для добавления персонализированных рекомендаций:

- найти хорошее место на странице;
- собрать и использовать список элементов, с которыми пользователь взаимодействовал;
- найти ассоциативные правила;
- сортировать по уверенности и вывести рекомендуемые элементы.

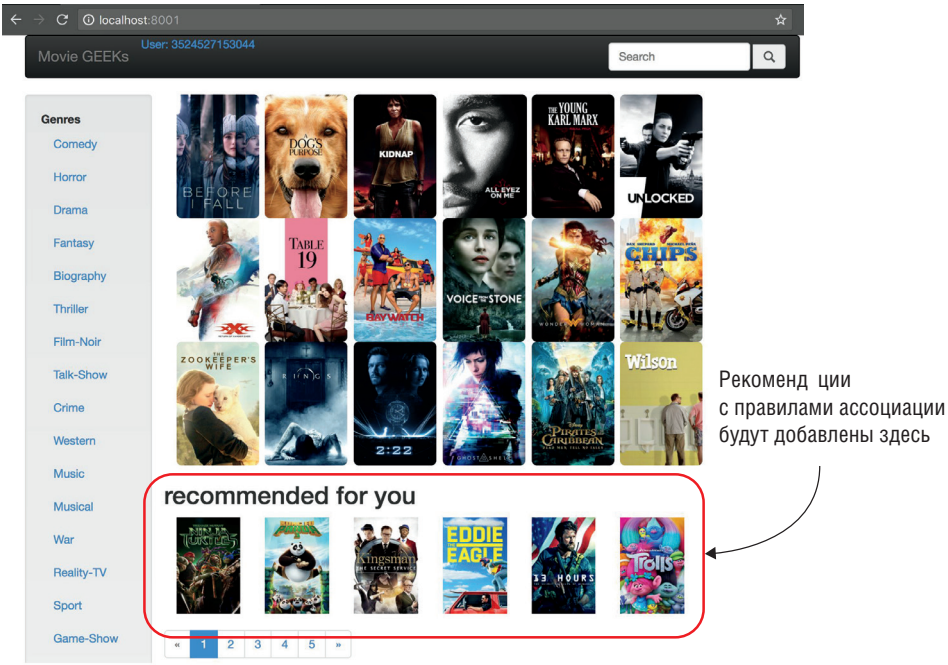


Рис 6.10. Добавленное пространство для персонализированных рекомендаций на сайте MovieGEEKs

6.5.1. Поиск собранных элементов

Вы хотите, чтобы эти рекомендации начали работать сразу же, как только пользователь будет что-то просматривать, так что вам стоит получить данные из журнала данных. Неявные оценки могут дать лучшее представление о предпочтениях пользователя, но, поскольку вам нужно обслуживать новых пользователей, вы не можете быть уверены в том, что система рассчитала неявные оценки.

Проблема, которую вы пытаетесь решить, – это создать рекомендации для пользователей, о ком вы знаете мало, так что вы будете использовать все элементы, с которыми пользователь взаимодействовал, а не только купленные. Данные немногочисленны, и в большинстве случаев пользователь не купил ничего до этого момента.

6.5.2. Получение ассоциативных правил и сортировка в соответствии со значениями уверенности

Вы можете использовать метод, который вы уже реализовали, чтобы получить ассоциативные правила, но это будет означать, что пришлось бы несколько раз делать запрос в базу данных. Вместо этого мы создадим новый метод, показаный в листинге 6.1, который делает запрос в базу данных самостоятельно.

ЛИСТИНГ 6.1. Расчет рекомендаций с использованием ассоциативных правил

```
def recs_using_association_rules(request, user_id, take=6):
    events = Log.objects.filter(user_id=user_id)\
                        .order_by('created')\
                        .values_list('content_id', flat=True)\
                        .distinct()

    seeds = set(events[:20])
    rules = SeededRecs.objects.filter(source__in=seeds) \
                              .exclude(target__in=seeds) \
                              .values('target') \
                              .annotate(confidence=Avg('confidence')) \
                              .order_by('-confidence')

    recs = [{'id': '{0:07d}'.format(int(rule['target'])),
             'confidence': rule['confidence']} for rule in rules]

    return JsonResponse(dict(data=list(recs[:take])))
```

З прос из б зы д нных событий, который связ н с идентифик тором пользов теля, сортировк по времени, созд ние и возвр т уник льного списк элементов

Берутся 20 с мых новых событий

З прос ссоци тивных пр вил и поиск всех пр вил, где н йдены д нные в контенте и журн ле пользов теля

Отобр жение целей вне журн л событий пользов теля

Берется только целев я строк

JSON и возвр т

Созд ние слов ря результ т

Сортировк по среднему зн чению уверенности

Дублирует цели в результ те, берется среднее зн чение уверенности

Вызов метода в листинге 6.2 дает вывод в формате JSON. Чтобы проверить это, попробуйте запросить **http://moviegeek.com:8000/rec/ar/5/**. Он производит JSON, который выглядит следующим образом:

ЛИСТИНГ 6.2. Результат запроса http://moviegeek.com:8000/rec/ar/5/

```
{
data:
[
{confidence: 0.006463878326996198,
id: "1291150"},
{confidence: 0.004617055947854427,
id: "1985949"},
{confidence: 0.004562737642585552,
id: "2267968"},
{confidence: 0.004562737642585552,
id: "0475290"},
...
}
```

Зн чения могут отлич ться из-з изменений в д нных

Ответ содержит больше элементов

6.5.3. Отображение рекомендаций

Если вы создаете новую вкладку браузера (на ПК в браузере Chrome это делается нажатием сочетания клавиш **Ctrl+Shift+N**), вы создадите новую сессию (рис. 6.11). Таким образом, можно имитировать нового пользователя. *Приватная вкладка* означает, что вы скрываете все текущие куки (и удаляете новые, когда покидаете приватную вкладку). Это полезно в таких случаях и позволяет увидеть, как выглядит сайт, когда вы зайдете в качестве нового пользователя.

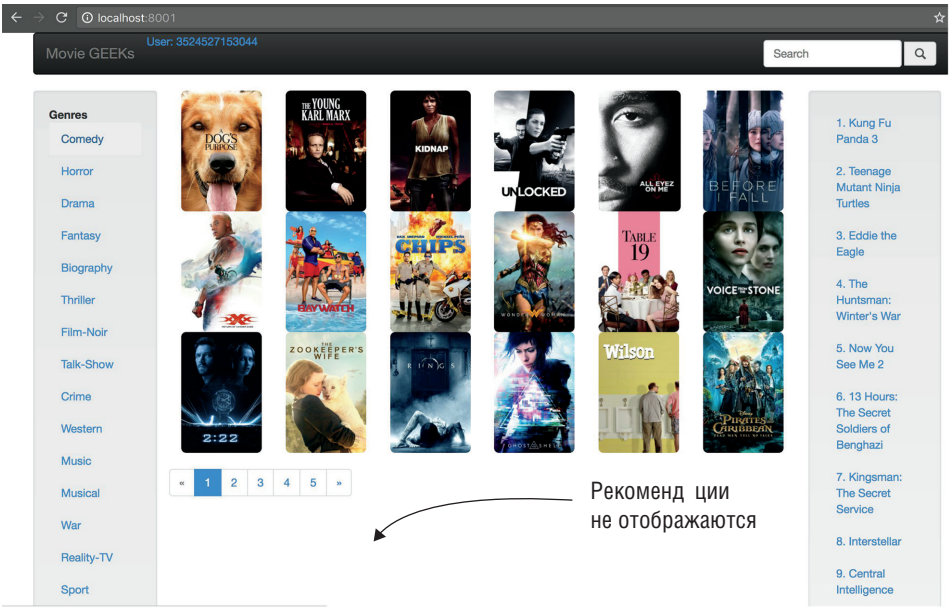


Рис. 6.11. Новая приватная сессия показывает, что новые посетители видят на сайте MovieGEEKs

Приватная вкладка позволяет просматривать сайт MovieGEEKs, как если бы вы посетили его в первый раз.

В окне консоли, где вы работаете с Django (рис. 6.12), понятно, почему рекомендации не отображаются. Поскольку это новая приватная сессия, она считается новым пользователем и сайт не распознает пользователя. Далее на рис. 6.12 видно, что алгоритм не находит сидов, т. е. данных нет.

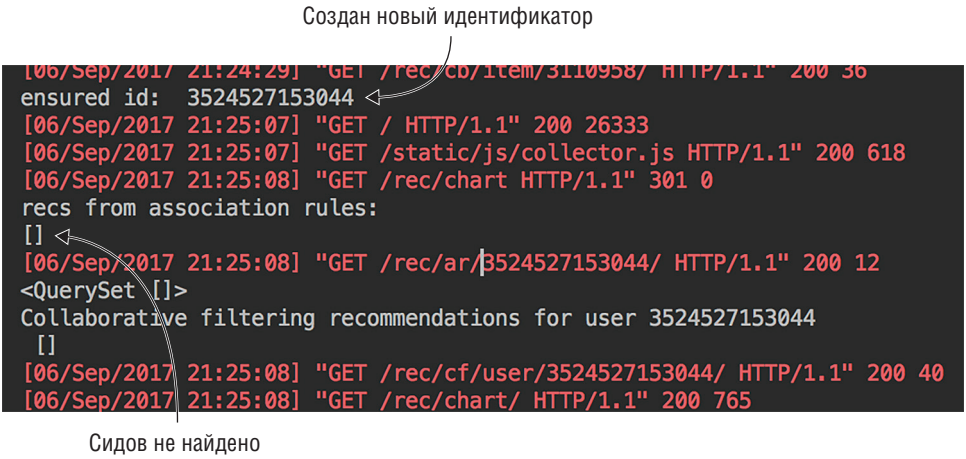


Рис. 6.12. Командная строка показывает новый идентификатор пользователя.

Давайте купим товар (**http://localhost:8000/movies/movie/3110958/**). Нажмите кнопку покупки и убедитесь, что ваш журнал приложения Django показывает зарегистрированный элемент данных. Это приведет вас на страницу, показанную на рис. 6.13.

Возвращаясь к главной странице, вы должны увидеть те же рекомендации, как показано на рис. 6.14, хотя порядок пленок отличается. (Следует отметить, что эти рекомендации имеют следующие идентификаторы контента: 10, 18, 45 и 9.)

Теперь, чтобы добавить интереса, давайте купим другой элемент в списке ассоциативных правил так, что рекомендации будут обновлены. Тут есть небольшой трюк. Если вы купили самый популярный элемент, уровень уверенности уже используемых правил будет гораздо больше, чем у других, поэтому, прежде чем начать бросать что-то из окна, если это ничего не меняет, посмотрите на значение уверенности.

Посмотрев на предыдущие ассоциативные правила, вы сможете увидеть, что если вы выбрали *Y*, то по крайней мере один элемент должен измениться. Перейдите по ссылке **http://moviegeeks:8000/movies/Y** и нажмите кнопку покупки. Затем вернитесь на главное окно и нажмите кнопку обновления (**F5** на большинстве компьютеров и браузеров). Рекомендации должны быть обновлены.

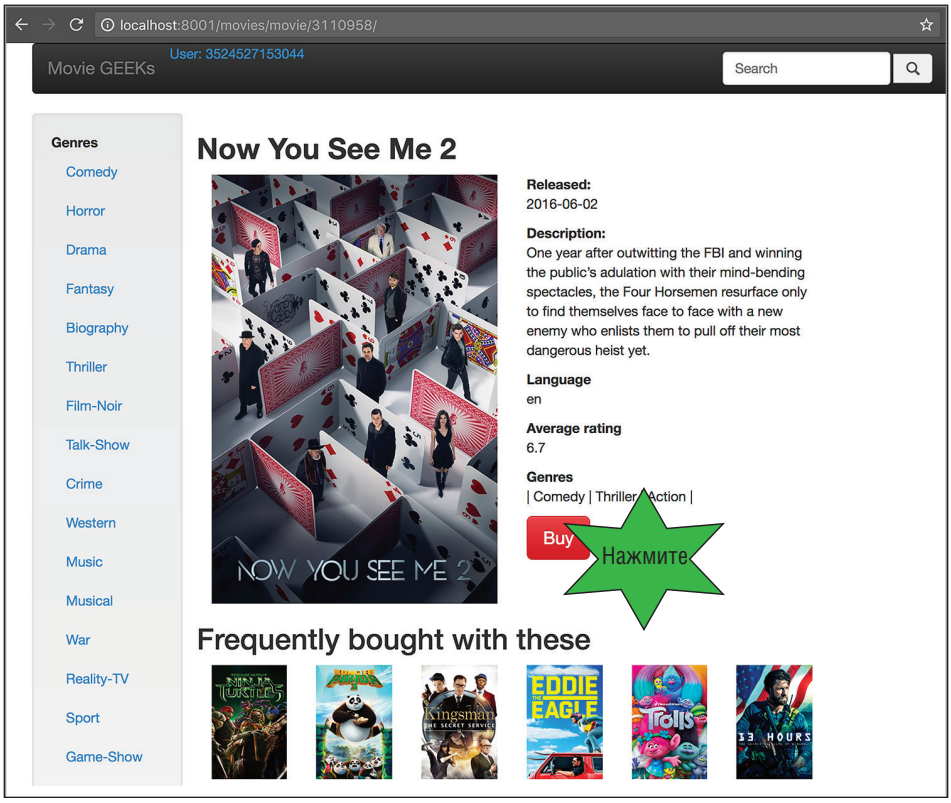


Рис. 6.13. Покупка «Иллюзии обмана – 2»

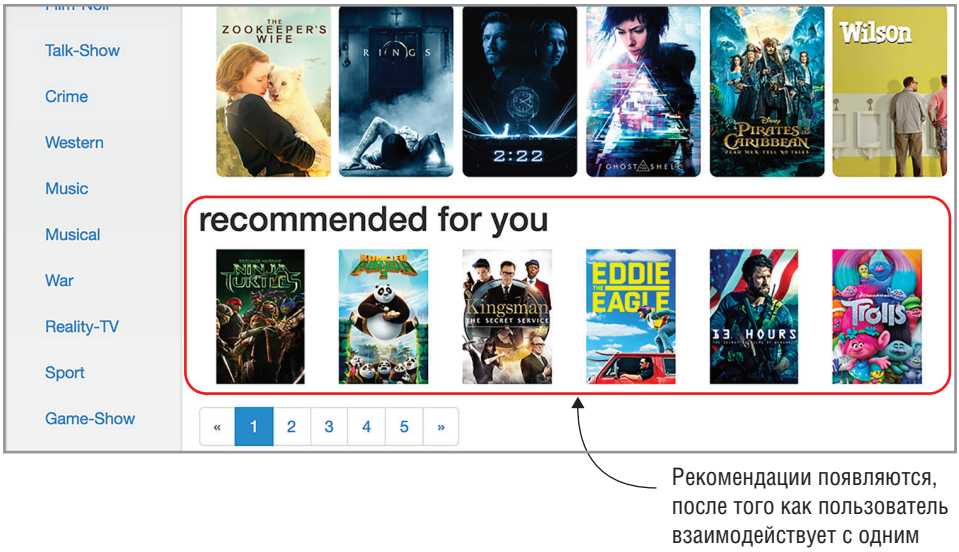


Рис. 6.14. Рекомендации после взаимодействия с одним элементом

6.5.4. Оценка реализации

Рекомендации получились простыми, но преимущество тут в том, что вы начнете получать их сразу. Когда вы соберете больше данных, у вас появятся более эффективные способы извлечения пользовательских настроек, которые мы рассмотрим в следующих главах.

Стоит также помнить, что данные, на которых вы основываете рекомендации, автоматически сгенерированы для MovieGEEKs. Генерация учитывает только жанры, так что ассоциации вроде «Ниндзи-убийцы» и «Жены путешественника во времени» кажутся неправдоподобными, учитывая, что один фильм – боевик, а другой является драмой.

Мы рассмотрели много тем в этой главе. Рекомендательные системы охватывают множество проблем, которые могут быть атакованы во многих отношениях. Различные исследования показывают – один лучше, чем другие, но только для определенных наборов данных. Важно знать ваши варианты, а затем посмотреть, что работает лучше всего для вашего сайта. На этом закончим первую часть этой книги. Мы рассмотрели все основы рекомендательных систем и поговорили о пользователях и о том, как понять их. Теперь настало время заглянуть в алгоритмы и понять, как рекомендатор делает свою «магию». Как вы увидите, в этом на самом деле никакой магии, но крутости от этого меньше не становится.

Резюме

- Проблема холодного старта означает, что вы должны решить, что рекомендовать новым посетителям на вашем сайте. Введение новых продуктов также является проблемой холодного запуска, который мы рассмотрим более подробно в последующих главах.
- Добавление сознательного упорядочения данных – это отличный старт, он позволит сделать многие сайты более динамичными.
- Серые овцы – это пользователи, чьи вкусы чересчур не похожи на вкусы других пользователей. Иногда серые овцы могут помочь в абстрагировании контента и разделении на жанры.
- Создание рекомендаций для новых пользователей с помощью ассоциативных правил – это легкий и быстрый способ добавить персонализацию.
- Можно вручную или автоматически создать сегменты, чтобы получить полуперсонализированные рекомендации. Кроме того, можно использовать демографические данные.

ЧАСТЬ II

Рекомендательные алгоритмы

Чтобы поверить в алгоритм, нужно его увидеть.
Дональд Кнут

В первых шести главах вы узнали об экосистеме и инфраструктуре рекомендательных систем. Пора перейти к части II, где мы рассмотрим алгоритмы рекомендательных систем; изучим, как использовать собираемые системой данные для вычисления того, какие вещи система должна рекомендовать пользователю; также обсудим, как можно оценить работу рекомендательной системы; и выявим сильные и слабые стороны каждого алгоритма.

Глава 7

Выявление общих черт у пользователей и контента

Сходство можно рассчитать разными способами, и мы расмотрим большинство из них. Что вас ждет в этой главе:

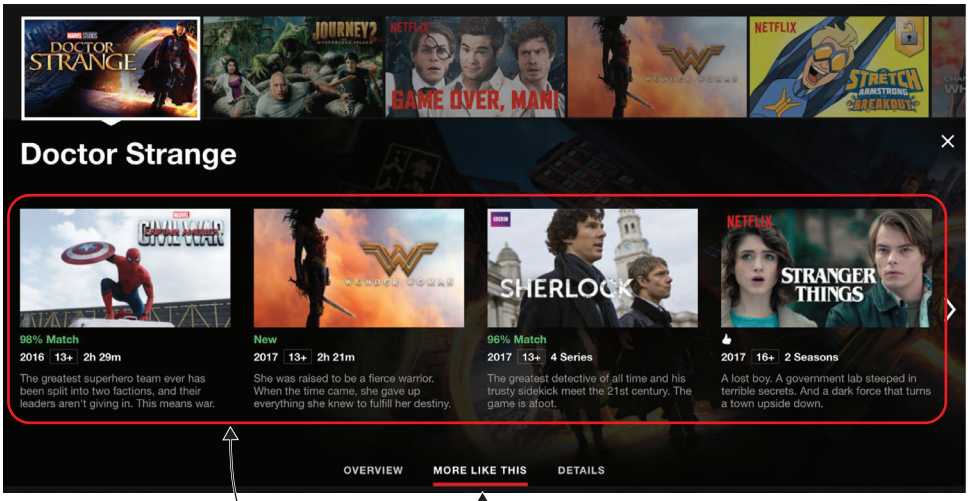
- общее понятие того, что такое сходство и его антипод – различие, или несходство;
- как вычислить сходство между наборами элементов;
- мы измерим сходство двух пользователей с помощью функций подобия, взяв за основу данные об их оценках контента;
- иногда бывает полезно группировать пользователей, и мы попробуем сделать это с помощью алгоритма кластеринга *k*-средних.

В главе 6 были описаны неперсонализированные рекомендации и ассоциативные правила. Ассоциативные правила – это такой способ связи контента независимо от элементов самого контента или пользователей, которые его потребляли. А вот персонализированные рекомендации почти всегда составляются на основе вычисления сходства. В качестве примера можно привести рекомендации More Like This на сайте Netflix. Эта рекомендация показана на рис. 7.1, а в ее построении используется алгоритм, который определяет похожий контент.

В следующих разделах мы узнаем, как рассчитать степень сходства двух элементов или двух пользователей. Сперва рассмотрим, как рассчитать сходство на бинарных данных (по-простому: купил или не купил товар), а затем двинемся дальше и попробуем вычислить сходство между пользователями на основе их оценок (и сходство элементов по оценкам одного и того же набора пользователей). Инструменты измерения сходства позволят вам хорошо понять, чем именно похожи те или иные элементы. В главе 8 мы воспользуемся этим, когда будем создавать алгоритмы совместной фильтрации.

В этой главе мы также узнаем, как можно сгруппировать пользователей на основе их предпочтений. Благодаря этому создадим функцию поиска для

пользователей. В главе 8 мы увидиим примеры того, как кластеризация позволяет оптимизировать алгоритмы совместной фильтрации. Эта глава, скорее, прикладная, но очень важная и для рекомендательных систем, и для многих алгоритмов машинного обучения. Но сперва давайте рассмотрим рекомендации, показанные на рис. 7.1.



На сайте Netflix есть раздел *More Like This* (похожие фильмы), который формируется, вероятно, путем вычисления подобия.

Рис. 7.1. Персональные рекомендации на сайте Netflix, построенные для любителя сериала «Флеш»

Из рисунка ясно, что речь идет не только о том, чтобы вывести фильмы, похожие на выбранный фильм «Доктор Стрэндж». Если бы мы хотели занести фильм «Доктор Стрэндж» в какой-нибудь каталог, я бы отнес его к категории **Супергерои**, куда также попал бы фильм «Первый Мститель: Противостояние» и «Чудо-женщина». А что насчет двух других фильмов? Как сюда попали они?

В этой главе мы рассмотрим вычисление сходства среди контента и среди пользователей с помощью различных метрик. Начнем, пожалуй, с самого понятия «сходство» и его вычисления.

7.1. Что за сходство?

Само это понятие нам нужно, для того чтобы научиться определять похожие на ваши предпочтения элементы или пользователей, которым нравится то же, что и вам. Так что же такое *сходство*? Рассмотрим пример: пусть у нас есть два человека. Как можно определить, насколько они похожи, по шкале от –1 до 1?

Наверное, логично спросить: «В каком смысле похожи?» Давайте добавим конкретики – пусть будет сходство во вкусах. Но и тут ответ может быть самым разным. Можно было бы сказать, что у двух людей схожие вкусы, если им обоим

нравятся фильмы с Томом Хэнксом, фантастические фильмы или просто фильмы «на вечер»[1]. Но даже у любителей фантастики могут быть разные вкусы. Кому-то нравится «Стартрек», а кому-то – «Звездные Войны». Похожи ли они?

Имеющиеся у вас данные позволяют сузить возможности использования оценок, чтобы понять вкус пользователя. Но сходство также можно определить с помощью дополнительной информации, например метаданных контента или данных о самом пользователе. Здесь мы узнаем, как же все-таки определить, похожи ли два элемента или нет. В научной литературе есть несколько функций подобия, которые, как говорят, дают хорошие результаты, и далее мы рассмотрим их все. Также поговорим об оптимизации количества вычислений сходства пользователей путем нахождения сегментов похожих пользователей.

7.1.1. Что такое функция подобия?

Подобие можно рассчитать по-разному, но в общем и целом задача может быть определена следующим образом: имеется два элемента, i_1 и i_2; сходство между ними определяется функцией sim (i_1, i_2).

Возвращаемое этой функцией значение пропорционально степени сходства между элементами. Тогда для идентичных элементов sim (i_1, i_1) = 1, а для элементов, не имеющих ничего общего, sim (i_1, i_2) = 0. На рис. 7.2 показаны примеры двух функций сходства. Выбирать из них нужно ту, которая больше подходит для вашего домена и ваших данных.

Измерение сходства тесно связано с расчетом несходства между элементами. Как правило, сходство и несходство соотносятся следующим образом:

- с увеличением несходства сходство стремится к нулю;
- с уменьшением несходства сходство стремится к единице.

7.2. Основные функции подобия

Как уже упоминалось ранее, универсально правильного или неправильного метода решения этой задачи не существует. Разные методы работают лучше на разных наборах данных, однако есть несколько общих принципов, которые мы обсудим в этом разделе.

Сперва рассмотрим *коэффициент сходства Жаккара*, который используется для сравнения наборов данных. В нашем случае этим набором будет множество фильмов, которым пользователь поставил оценку. Затем мы посмотрим на сходство между оценками – сходство между оценками одного фильма, поставленными двумя пользователями. Эти данные позволят оценить, насколько схожи два пользователя, если они оценят достаточно много фильмов. Для этого воспользуемся функциями сходства Пирсона и косинусной функцией. Для каждого метода сходства необходим определенный вид данных, как показано в табл. 7.1.

[1] У нас в Дании есть термин *helaftensfilm*, под которым понимается более длинный фильм по сравнению с нормальными полнометражными фильмами. Это, как правило, фильм длиннее 2 ч 45 мин.

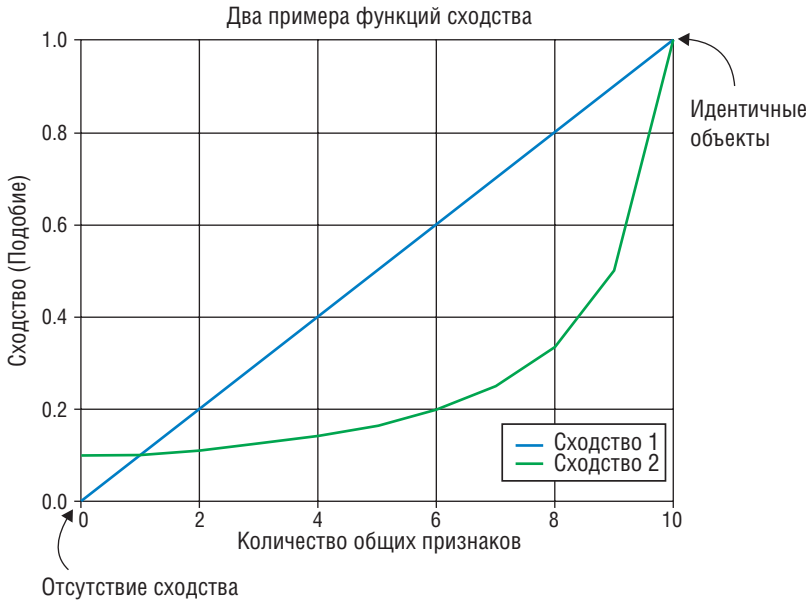


Рис. 7.2. Две различные функции сходства. Прямая линия означает, что сходство равно 0,5, если половина признаков элементов совпадает. Кривая линия дает подобие приблизительно 0,2

Таблица 7.1. Различные типы данных

Тип данных	Пример данных	Подобие
Унарные данные: это может быть, например, количество лайков или покупок	Пользователю 1 понравился фильм 2. Пользователю 2 понравился фильм 2. Пользователю 3 понравился фильм 1	Сходство Жаккара
Бинарные данные: данные, у которых есть два возможных значения, например «нравится-не нравится»	Пользователю 1 не понравился фильм 1. Пользователю 1 понравился фильм 2. Пользователю 2 понравился фильм 2. Пользователю 3 понравился фильм 1	Сходство Жаккара
Количественные данные	Пользователь 1 поставил 4/10 звезд фильму 2. Пользователь 2 поставил 10/10 звезд фильму 2. Пользователь 3 поставил 1/10 звезд фильму 1	Функция Пирсона, или косинус

Перед тем как начать, в табл. 7.2 мы перечислим несколько вещей, которые позволяют легко описать функцию сходства.

Таблица 7.2. Элементы функции сходства

Название	Определение	Пример
r_{ui}	Оценка элемента *i* пользователем *u*	$R_{сара,\ стартрек}$ = 4,5 – Сара поставила оценку 4,5 фильму «Стартрек»
r_u	Средние оценки пользователя *u*	Среднее из всех оценок, поставленных пользователем *u*. Если Питер поставил 4 и 3 фильмам «Звездные Войны» и «Стартрек», то $r_{питер}$ = 3,5
$P_{a,b}$	Набор элементов, которым поставили оценку пользователи *a* и *b*	Набор элементов с Сарой и Питером из предыдущих примеров. $P_{сара,питер}$ = *{стартрек}*, если они оба поставили оценку фильму

7.2.1. Расстояние Жаккара

Изначально этот параметр назывался *коэффициентом сходства Жаккара*, который показывает, насколько схожи два набора данных. Иногда он называется «индексом Жаккара» или «коэффициентом сходства Жаккара».

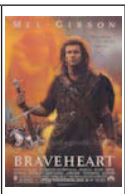
Стоп, два набора? И как это касается пользователей и контента? Допустим, если каждый фильм – это такой список пользователей, которые купили этот фильм, то у вас есть по одному набору пользователей для каждого фильма. Тогда мы можем сравнить два фильма, взяв две такие группы пользователей.

Данные наборы, такие как описанные выше, можно получить из пользовательских покупок, превращенных в список, где в каждой строке написано, купил или не купил пользователь продукт (1 = купил, 0 = не купил). Если пользователю понравился контент, у вас есть унарный набор данных, который можно свести в таблицу, как, например, в табл. 7.3. Это унарный набор данных, где 1 означает наличие данных, а 0 – их отсутствие.

Таблица 7.3. Унарная матрица пользователей и элементов (1 = купил, 0 = не купил)

						
	Комедия	**Экшн**	**Комедия**	**Экшн**	**Драма**	**Драма**
Сара	1	1	0	0	0	0
Джаспер	1	1	1	0	0	0
Тереза	1	0	0	0	0	0
Хелла	0	1	0	1	0	0

Окончание табл. 7.3

						
	Комедия	**Экшн**	**Комедия**	**Экшн**	**Драма**	**Драма**
Петро	0	0	0	0	1	1
Екатерина	0	0	0	0	1	1

Чтобы найти сходство между двумя элементами, нужно подсчитать, сколько пользователей купили оба элемента, а затем разделить на количество пользователей, которые купили один из них (или оба). В виде формулы это будет выглядеть так:

$$сходство_{Жаккара}(i,j) = \frac{\#пользователи,\ купившие\ оба\ товара}{\#пользователи, которые\ купили\ либо\ i, либо\ j},$$

где i – это элемент 1 и j – элемент 2.

Чтобы вычислить сходство между двумя фильмами, нужно подсчитать количество одинаковых битов (сколько раз пользоватлеи сделали с каждым фильмом одно и то же действие), как показано в табл. 7.4. В таблице даны четыре ряда из шести, в которых пользователи сделали то же самое, т. е. сходство Жаккара между ними равняется 4/6. Если вы посмотрите на фильмы в табл. 7.4, за которые пользователи не голосовали, сходство Жаккара у них равно 2/4 = 0,5, т. е. у них есть небольшое сходство.

Таблица 7.4. Сходство между фильмами «Люди в черном» и «Стартрек»

			
Сара	1	1	1
Джаспер	1	1	1
Тереза	1	0	0
Хелла	0	1	0
Петро	0	0	1
Екатерина	0	0	1
			Сумма = 4

Сходство Жаккара показано на предыдущем рисунке как сходство 1 (см. рис. 7.2). Сходство 0,5 может быть высоким для вашего случая, но лучше попробовать разные функции сходства и оценить, какая лучше подходит для данной задачи. Позже мы рассмотрим сходство пользователей на сайте MovieGEEKs, используя алгоритм сходства Жаккара. Сейчас, если в вашем наборе данных есть более подробная информация, вы можете выполнить дополнительные вычисления сходства. Давайте рассмотрим некоторые из них.

7.2.2. Измерение расстояния с помощью L_p-норм

Рассмотрим измерение расстояния, или несходства, с помощью L_p-норм и попробуем две различные меры, самые L_1- и L_2-нормы. Если набор данных является более детализированным, чем в примере MovieGEEKs, и известно, скольким людям понравился контент, вы можете использовать множество других функций для расчета расстояния и сходства, а не только меру Жаккара.

L_1-норма

Что такое сходство? Мы уже поднимали этот вопрос, но сейчас, надеюсь, ответим на него. Если нам нужно узнать, сходятся ли мнения Петро и Сары о фильме «Тайная жизнь домашних животных», можно попросить их оценить фильм по шкале от 1 до 10. Петро этот фильм показался неплохим, так что он поставил 6, а вот Сара обожает и собак, и мультики, поэтому поставила 9, и это был почти идеальный фильм для нее. На рис. 7.3 показаны оценки Петро и Сары.

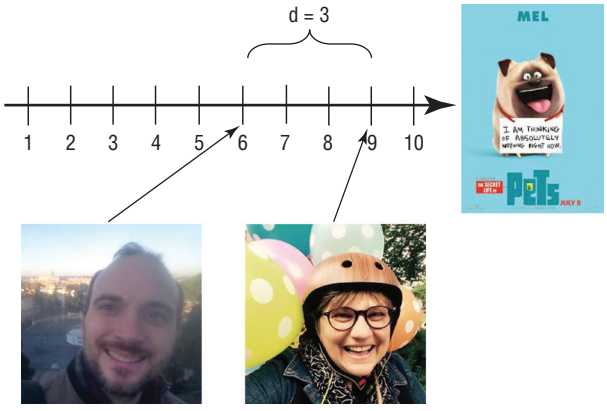


Рис. 7.3. Сходство оценок двух пользователей для одного фильма

На основе оценок одного фильма легко измерить сходство двух пользователей, даже если это не дает каких-либо реальных признаков их реальных вкусов или предпочтений. Вы можете рассчитать разницу между пользователями следующим образом:

$$расстояние(сара, петро) = \left| r_{сара} - r_{петро} \right|,$$

где $r_{сара}$ – оценка Сары; $r_{петро}$ – оценка Петро.

По этой формуле расстояние равняется 9 – 6 = 3. Поскольку максимальное расстояние между двумя оценками составляет 9, вы можете рассчитать сходство следующим образом (где 1 соответсвует минимальному расстоянию, а 0 – максимальному):

$$сходство(сара, петро) = \frac{1}{|r_{сара} - r_{петро}| + 1}.$$

Здесь я добавил 1 к знаменателю, чтобы избежать деления на ноль, в случае если оценки будут равны. Вернемся к расстоянию – для двух фильмов, как показано на рис. 7.4, вы можете вычислить разницы оценок и сложить их, получив следующее:

$$абсолютная\ сумма\ различий\ (SAD) = \sum_{i=1}^{n} |r_{сара,i} - r_{петро,i}|.$$

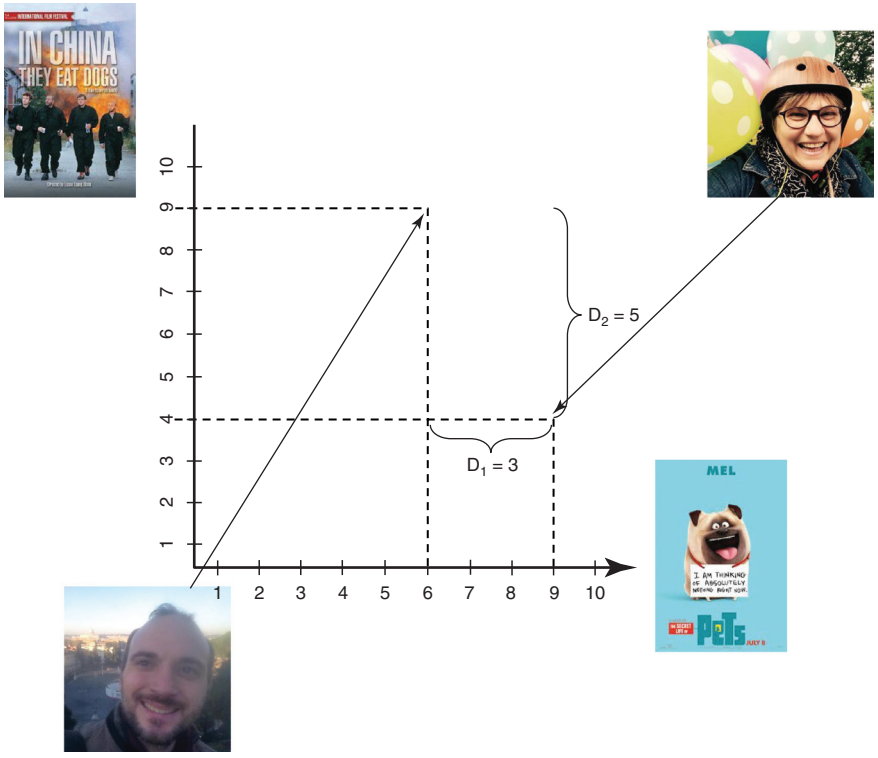


Рис. 7.4. Сходство двух пользователей на основе оценок двух фильмов. Фильму «Быстрые стволы» Петро поставил 9, а Сара – 4, поэтому разница D2 = 5, а у «Тайной жизни домашних животных» разница D1 = 3

Такой способ измерения расстояния или сходства называется «Расстояние Манхэттена» и является частью так называемой метрики *Расстояния город-*

ских кварталов[1]. Более официально она называется L_1-нормой. Ее идея заключается в том, что для измерения расстояния между двумя углами улиц в Нью-Йорке требуется сетка, а не прямые линии.

По L_1-норме сходство будет равно 3 + 5 = 8. Часто требуется найти среднюю абсолютную ошибку (MAE), которая рассчитывается через среднее значение L_1-нормы. Как видно из следующей формулы, новым здесь является то, что вы делите сумму на количество элементов, чтобы получить среднее расстояние между оценками:

$$\textit{средняя абсолютная ошибка}\left(MAE\right) = \frac{1}{n}\sum_{i=1}^{n}\left|r_{\textit{сара},i} - r_{\textit{петро},i}\right|.$$

Мы вернемся к MAE в главе 9, когда будем оценивать работу рекомендательных систем. Возвращаясь к теме сходства, давайте перейдем к следующей норме.

L_2-норма

У L_1-нормы есть старшая сестра – L_2-норма, которая геометрически соответствует расстоянию между двумя точками в Нью-Йорке, но уже не для такси, а, например, для вороны – т. е. напрямую. Тут нужно вспомнить теорему Пифагора, $a^2 + b^2 = c^2$, которая говорит, что квадрат гипотенузы (сторона, лежащая напротив прямого угла в треугольнике) равна сумме квадратов двух других сторон. Если вы не слышали о Пифагоре или его теореме, не переживайте – мы уже двигаемся дальше.

L_2-норма иначе называется *евклидовой нормой* и определяется следующим образом:

$$\textit{расстояние}\left(\textit{сара},\ \textit{петро}\right) = r_{\textit{сара}} - r_{\textit{петро}2} = \sqrt{\sum_{i=1}^{n}\left|r_{\textit{сара},i} - r_{\textit{петро},i}\right|^2}.$$

Работая с машинным обучением, вы непременно столкнетесь с евклидовой нормой. Она часто используется в качестве меры того, насколько хорошо работает ваш алгоритм. Она берет среднее нормы под названием *средняя квадратичная ошибка* (RMSE), которая тоже используется и тут и там:

$$\textit{средняя квадратичная ошибка}\left(\text{RMSE}\right) = \frac{1}{n}\sum_{i=1}^{n}\left|r_{\textit{сара},i} - r_{\textit{петро},i}\right|^2.$$

Опять же, можно перейти к сходству, добавив 1 над суммой квадратов разностей. Этими формулами, в принципе, можно пользоваться, но для рекомендательных систем это все-таки не лучшее решение. Но мы их упомянули, так как будем использовать их при оценке алгоритмов. В следующем разделе обсудим более качественный способ измерения сходства.

---

[1] Более подробно: **ru.wikipedia.org/wiki/Расстояние_городских_кварталов**.

7.2.3. Коэффициент Отиаи

Можно рассматривать контент следующим образом: берем строки матрицы оценок как векторы в пространстве и измеряем угол между ними. Это звучит слегка «пространно», но все станет понятнее на следующем примере, где мы сведем ваши данные в упрощенном виде в табл. 7.5.

Таблица 7.5. Упрощенная матрица оценок

		
	Комедия	**Экшн**
Сара	3	5
Тереза	4	1
Хелла	2	5

Построим эти данные в системе координат, показанной на рис. 7.5. Это работает для любого числа элементов, но мы остановимся на двух, так как нормальных способов изобразить большее количество измерений нет. Просто оговоримся, что метод применим к любому числу измерений.

Из углов между векторами легко видеть, что Сара и Хелла по своим оценкам гораздо ближе друг к другу, чем Тереза и Сара. То есть можно предположить, что вкусы Хеллы и Сары более схожи.

Тут есть небольшая проблема, которая заставляет долго смотреть на векторы. Если вы посмотрите на рис. 7.5 и представите, что один пользователь поставил фильму оценку 6, а другой – оценку 1, то векторы будут направлены в одну сторону. Таким образом, их сходство, по идее, должно равняться 1. Это может быть проблемой при вычислении сходства на основе углов между векторами; но на практике я с такой ситуацией ни разу не сталкивался.

Давайте опробуем эту методику, вычислив сходство разок-другой. Но применим это не к пользователям, а к элементам. Amazon своим примером показал нам, что совместная фильтрация типа «элемент–элемент» является гораздо более эффективной по той простой причине, что покупателей больше, чем товаров. А еще при вычислении сходства элементов требуется меньше расчетов.

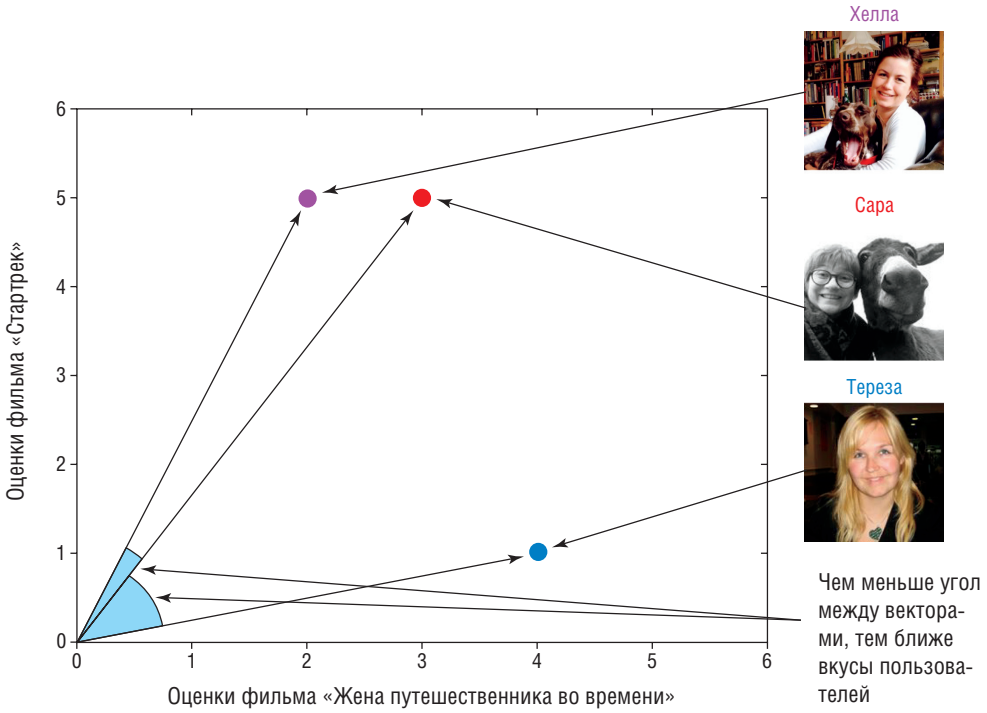


Рис 7.5. Можно измерить сходство, рассчитав углы между векторами оценок

Вычисление подобия элементов

Оценка сходства означает, что теперь мы будем смотреть на столбцы (см. табл. 7.6) вместо строк, как в предыдущем разделе.

Таблица 7.6. Цель пункта сходства

						
	Комедия	**Экшн**	**Комедия**	**Экшн**	**Драма**	**Драма**
Сара	5	5		2	2	2
Джаспер	4	5	4		3	3
Тереза	5	3	5	2	1	1
Хелла	3		3	5	1	1
Петро	3	3	3	2	4	5
Екатерина	2	3	2	3	5	5

Функция для вычисления угла между столбцами оценок реализуется с использованием формулы косинуса, которую все дети изучают в школе и благополучно забывают после нее. Так вот, если вы забыли формулу, напоминаем:

$$sim(i,j) = \frac{r_i \times r_j}{r_{12} r_{j2}} = \frac{\sum_u r_{i,u} r_{j,u}}{\sqrt{\sum_u r_{i,u}^2} \sqrt{\sum_u r_{j,u}^2}}.$$

Прекрасно, не так ли? К сожалению, нам придется слегка изменить функцию, потому что мы говорим о сравнении оценок различных пользователей, а люди используют разные шкалы оценок (радостный оценщик ставит более высокие оценки, чем грустный). Нужно принять это во внимание. Бадрул Сарвар и его друзья придумали скорректированный коэффициент Отиаи, в котором этот недостаток скомпенсирован путем вычитания средней оценки пользователя[1]. К счастью, формула все еще красивая:

$$sim(i,j) = \frac{\sum_u \left(r_{i,u} - \overline{r}_u\right)\left(r_{j,u} - \overline{r}_u\right)}{\sqrt{\sum_u \left(r_{i,u} - \overline{r}_u\right)^2} \sqrt{\sum_u \left(r_{j,u} - \overline{r}_u\right)^2}}.$$

7.2.4. Вычисление сходства с помощью коэффициента корреляции Пирсона

Если вы посмотрите на оценки, которые мы рассматривали в последних разделах, вы увидите, у кого есть сходство. Но если вы построите их на графике, где фильмы будут отложены по оси *x*, а оценки – вдоль оси *y*, а затем соедините их линией, – все станет еще более наглядно и будет видно, какие оценки схожи, а какие нет (рис. 7.6).

Коэффициент корреляции Пирсона берет эти точки и вычисляет, как сильно они отличаются от среднего: большое различие дает значение коэффициента, близкое к –1, а сходство дает коэффициент, близкий к 1. Следует отметить, что здесь результаты лежат в диапазоне от –1 до 1, а не 0 до 1, как вы видели ранее в табл. 7.4.

Алгоритм вычисляет, как сильно две линии двух пользователей коррелируют между собой. Если их вкусы одинаковы (линии идут вверх и вниз вместе), коэффициент будет равен 1 или близко к этому. Значение 0 означает отсутствие корреляции, а значение –1 – что вкусы пользователей совершенно противоположны.

[1] Бадруд Сарвар и др., «Рекомендательные алгоритмы совместной фильтрации по элементам» (**files.grouplens.org/papers/www10_sarwar.pdf**).

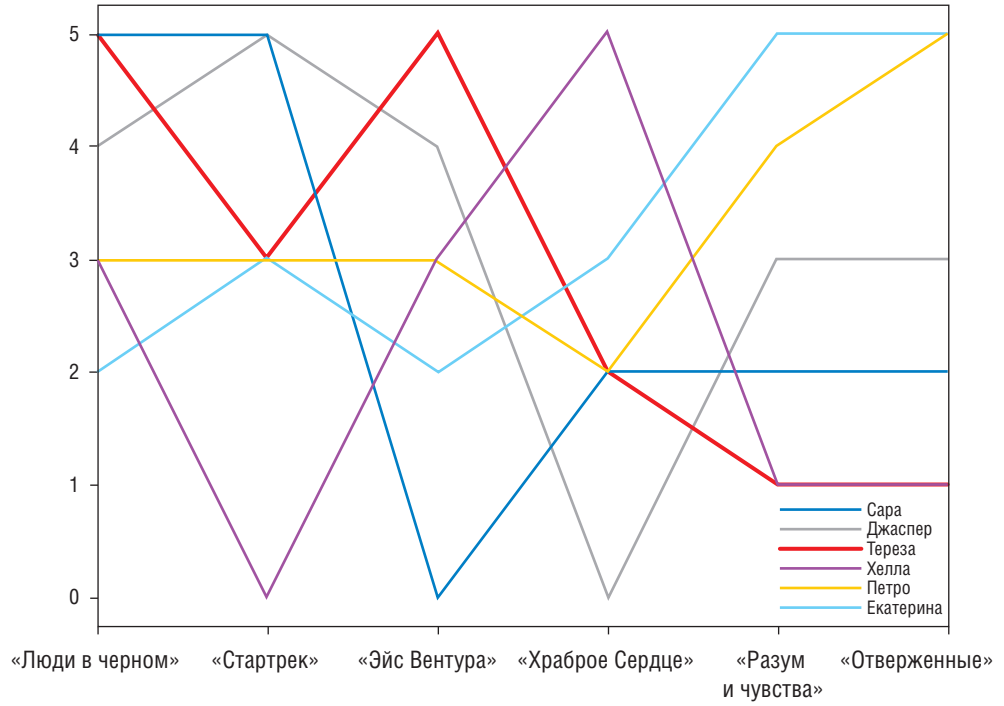


Рис. 7.6. Графики оценок пользователей MovieGEEKs. По вертикальной оси отложены оценки, а по горизонтальной – список фильмов. Фильмы перечислены в табл. 7.6: «Люди в черном», «Стартрек», «Эйс Вентура», «Храброе Сердце», «Разум и чувства», «Отверженные»

При использовании коэффициента корреляции Пирсона оценки нормируются путем вычитания средней оценки элемента из каждой оценки, как показано в следующем уравнении:

$$sim(i,j) = \frac{\sum_{e \in U}\left(r_{i,u} - \overline{r}_i\right)\left(r_{j,u} - \overline{r}_j\right)}{\sqrt{\sum_{e \in U}\left(r_{i,u} - \overline{r}_i\right)^2}\sqrt{\sum_{e \in U}\left(r_{j,u} - \overline{r}_j\right)^2}},$$

где i и j – это множества поставивших оценки пользователей.

Все выглядит немного страшно, но на самом деле это не так. Тут просто сложение и умножение, и далее мы посмотрим, что все легко реализуется. Давайте попробуем узнать, насколько схожи вкусы у Джаспера и Петро.

7.2.5. Испытание сходства коэффициентом Пирсона

В табл. 7.7 показаны оценки Джаспера и Петро для шести фильмов.

Таблица 7.7. Оценки Джаспера и Петро

						
	Комедия	**Экшн**	**Комедия**	**Экшн**	**Драма**	**Драма**
Джаспер	4	5	4		3	3
Петро	3	3	3	2	4	5

Для вычисления коэффициента Пирсона нужно:

1. Рассчитать средние оценки.
2. Нормализовать оценки.
3. Поместить результаты в формулу.

Расчет средней оценки

Во-первых, вам необходимо рассчитать среднюю оценку для каждого пользователя. Для этого надо сложить все оценки и разделить их на количество оценок:

- Джаспер: (4 + 5 + 4 + 3 + 3) / 5 = 3,8;
- Петро: (3 + 3 + 3 + 2 + 4 + 5) / 6 = 3,33.

Обратите внимание на то, что Джаспер оценил пять фильмов, а Петро оценил все шесть. Чтобы вычислить среднее значение, нам нужно использовать количество оценок, поставленных каждым пользователем. Затем среднее вычитается, чтобы оценки были сопоставимы друг с другом.

Поскольку Джаспер ставил оценки от 3 и 5, можно сделать вывод о том, что оценка 3 для него считается низкой, а оценка 5 – довольно высокой. Но если бы он поставил десяти другим фильмам 1, его средняя оценка была бы намного ниже, и тогда 3 выглядела бы более позитивно. Это также может означать, что Джасперу толком не понравился ни один фильм, но, чтобы наш расчет подобия сработал, придется исходить из оценок 3 и 4.

Примечание по реализации

Помните, что, когда у вас есть массив оценок, подобных оценкам Джаспера, вам необходимо рассчитать среднее значение для оцениваемых элементов. Оценки Джаспера дают нам массив [4, 3, 0, 4, 4]. Рассчитывая среднее значение стандартным способом, мы получим 15 / 5. Хотя ноль не влияет на сумму (15), он все равно входит в общее число оценок (5).

Нормализация оценок

Как упоминалось ранее, оценки определенного пользователя всегда следует рассматривать совместно с другими оценками этого пользователя. Для сравнения оценок двух пользователей необходимо нормализовать их. Понятие *нормализации* довольно широкое, но в своей простейшей форме оно означает, что шкала значений одного массива подгоняется под другие, чтобы они были сопоставимы.

Чтобы нормализовать оценки, нужно вычесть среднюю оценку каждого пользователя из их оценок. Например, чтобы вычесть среднее из оценок Джаспера и Петро, нужно вычислить $r_{джаспер,\,i} - \overline{r}_{джаспер}$ и $r_{петро,\,i} - \overline{r}_{петро}$, как показано в табл. 7.8. После этого у вас будут основные элементы для вычисления сходства по методике Пирсона.

Таблица 7.8. Нормализованные оценки Петро и Джаспера

Джаспер	0,20	1,2	0,2		–0,8	–0,8
Петро	–0,33	–0,33	–0,33	–1,33	0,67	1,67

После нормализации значение около 0 дает оценке позитивный или негативный характер. Например, оценка –0,75 звучит менее приятно, чем 3. Вероятно, эту оценку следует рассматривать как отрицательную. Но в зависимости от отношения пользователя она тем не менее может означать как симпатию к фильму, так и неприязнь.

Сводим результаты в формулу

Пусть $nr_{a,\,i} = r_{a,\,i} - \overline{r}_a$ преобразует корреляцию Пирсона в следующий вид, но с нормализованными оценками:

$$sim(a,b) = \frac{\sum_{i \in P}\left(nr_{a,i}\right)\left(nr_{b,i}\right)}{\sqrt{\sum_{i \in P}\left(nr_{a,i}\right)^2}\sqrt{\sum_{i \in P}\left(nr_{b,i}\right)^2}}.$$

Подставив нормализованные оценки, вы получите следующее:

$$\frac{(0,2)(-0,33)+(1,2)(-0,33)+(0,2)(-0,33)+(-0,8)(0,67)+(-0,8)(1,67)}{\sqrt{(0,2)^2+(1,2)^2+(0,2)^2+(-0,8)^2+(-0,8)^2}\sqrt{(-0,33)^2+(-0,33)^2+(-0,33)+^2(0,67)^2+(1,67)^2}}.$$

Теперь осталось только посчитать (а точнее просто скопировать формулу в браузер, и там все посчитается само):

$$sim(джаспер,\ петро) = \frac{-2,4}{\sqrt{2,8} \times \sqrt{3,56}} = -0,76.$$

Результат показывает то, что мы и так видели: вкусы Джаспера и Петро совсем расходятся. Можно даже сказать, что они близки к противоположным.

7.2.6. Коэффициент корреляции Пирсона на коэффициент Отиаи

Коэффициент корреляции Пирсона и коэффициент Отиаи выглядят похожими, и, более того, далее вы увидите, что подобие Отиаи расширяется в скорректированную функцию подобия. Под корректировкой здесь понимается добавление нормализации оценок, и в итоге мы получим ту же функцию Пирсона. Единственное различие между ними заключается в том, что в функции Пирсона используются элементы, которые оценили два пользователя, а в функции Отиаи будут использоваться все элементы, оцененные каждым из них, а неоцененный элемент соответствует значению 0.

В предыдущем примере с Джаспером и Петро две функции подобия дали нам следующий результат:

$$sim_{коэффициент\ Отиаи}\left(джаспер,\ петро\right) = -0,62,$$

$$sim_{Пирсон}\left(джаспер,\ петро\right) = -0,75.$$

Следует заметить, что, если вы вычисляете сходство между пользователем, который поставил всего несколько оценок, и пользователем, который ставит их много, то коэффициент Отиаи даст подобие, близкое к нулю. Почему? Большое количество элементов, за которые голосовал только один пользователь, дает произведение оценок, равное нулю, и к сумме ничего не добавляется. А вот знаменатель увеличивается с каждой оценкой.

7.3. Кластеризация k-средних

Как вы увидите в следующей главе, вычисление сходства между пользователями или элементами является ахиллесовой пятой алгоритмов совместной фильтрации. Дело в том, что алгоритм должен знать, как сильно текущие пользователи схожи со всеми остальными пользователями и как сильно текущие элементы схожи со всеми остальными элементами в системе. Поэтому было бы неплохо разделить массив данных на более мелкие группы, чтобы затем вычислять сходство в группах с меньшим количеством пользователей.

На рис. 7.7 показано разделение данных на четыре группы. Учтите, что каждый раз, когда на ваш сайт заходит человек, вам рано или поздно потребуется сравнить всех пользователей в системе с активным пользователем. Если ваши данные поделены на кластеры, можете посмотреть, в какой группе находится пользователь, и вуаля – три четверти вычислений уже можно не выполнять.

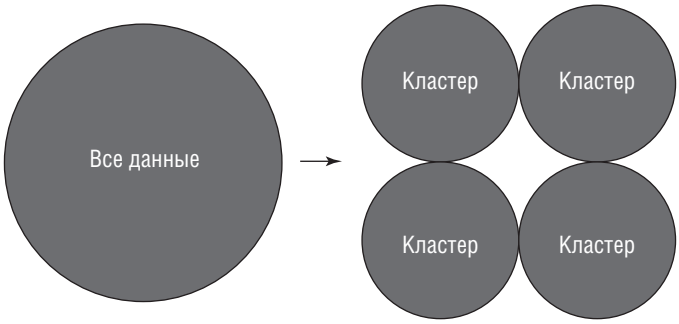


Рис. 7.7. Данные после кластеризации

7.3.1. Алгоритм кластеризации k-средних

Кластеризация также называется *сегментацией*. Мы уже затрагивали эту тему в предыдущей главе, когда говорили о неперсонализированных рекомендациях. Но тогда мы говорили о том, что сегментация используется с целью определить группы пользователей, подобных новому пользователю. Для вычисления сходства использовались демографические данные. Конечная цель всего этого – оптимизация, так как поиск кластеров пользователей позволяет уменьшить количество вычислений сходства пользователя.

Если к вам на сайт заходит пользователь *X*, вы, по идее, должны перебрать всех пользователей в базе данных. Но если ваши пользователя поделены в кластеры, то можно посмотреть, в какой кластер входит новый пользователь, и вычислить сходство между этими пользователями уже внутри кластера. Идея хорошая, но в ней есть риск того, что в кластере не окажется схожих пользователей. Тем не менее давайте опробуем кластеризацию на практике. Мы будем использовать алгоритм кластеризации k-средних – один из популярных алгоритмов сегментации.

Как работает кластеризация k-средних?

Кластеризация k-средних иначе называется *алгоритмом машинного обучения без учителя*. Приписка «без учителя» означает, что вы не даете алгоритму эталонных примеров того, как должны выглядеть правильные пары ввода-вывода. Этот алгоритм также является *параметрическим*, так как для его работы требуется параметр k. Добавить один параметр – звучит несложно, но вот правильно подобрать его значение трудно, а от этого, к сожалению, зависит правильное формирование групп пользователей.

Параметр k указывает алгоритму, сколько кластеров он должен выделить. В следующем разделе на примере простой реализации на Python вы узнаете, как работает кластеризация k-средних в решении вот такой задачи: требуется разделить пользователей (Сара, Дея, Питер, Мела, Ким, Хелла, Эголь, Влад и Джимми) на две группы, основываясь на оценках двух фильмов. Оценки такие:

- Сара = [7,1]
- Дея = [10,0]
- Питер = [0, 6]
- Мела = [1, 4]
- Ким = [5,3]
- Хелла = [9,9]
- Эгле = [2,1]
- Влад = [4,4]
- Джимми = [6,8]

Первых двух разделить легко, так как один фильм им понравился, а другой – нет, а вот с остальными посложнее. На рис. 7.8 показаны их оценки на графике.

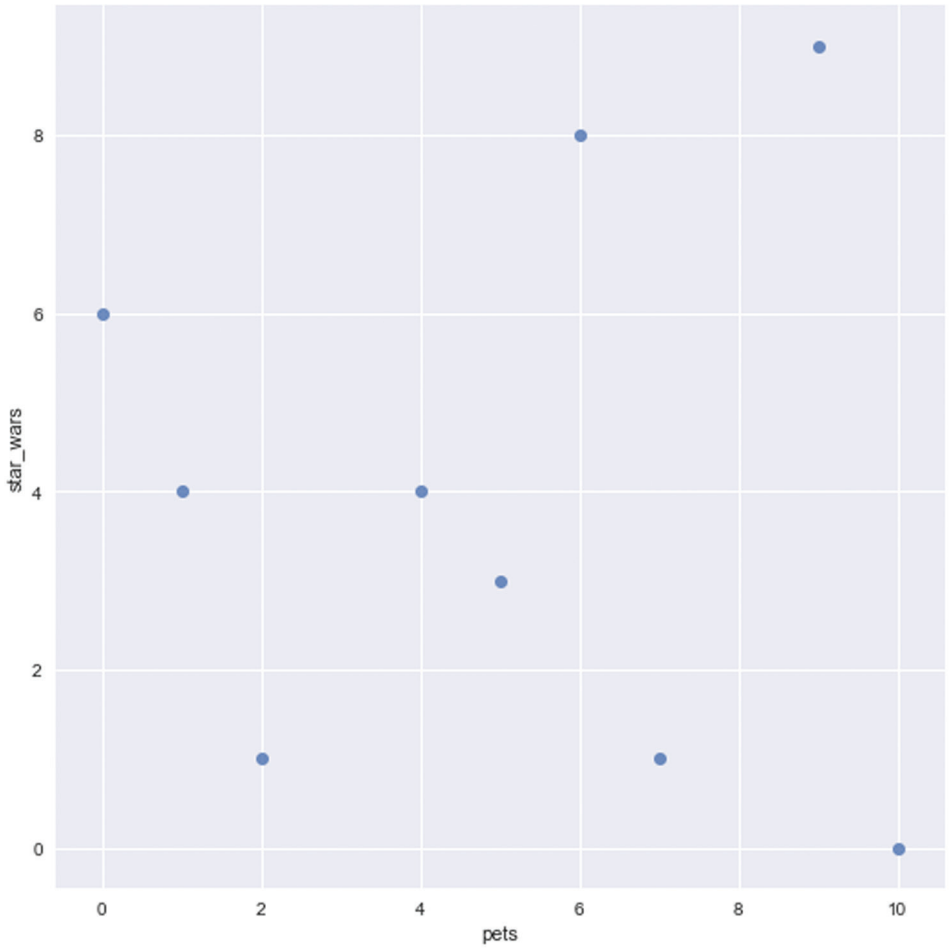


Рис. 7.8. Каждая точка – оценки пользователя для «Звездных Войн» и «Тайной жизни домашних животных»

Алгоритм кластеризации k-средних находит такие точки *k*, называемые *центроидами*, которые удовлетворяют утверждению о том, что сумма расстояний между всеми элементами и присвоенными ими центроидами будет настолько мала, насколько это возможно. Пока пишу этот текст, смотрю как на участке работает поливалка для цветов. Задача примерно похожая – надо крепко подумать о том, где разместить поливалку, потому что вода в Дании дорогая, и нужно найти такую точку, чтобы поливать как можно больше цветов, а воды тратить поменьше. Кластеризация k-средних подойдет и для этой задачи, так как этот алгоритм сгенерирует центроиды, в которые можно разместить поливалку.

Алгоритм выполняет следующие этапы:

- Выбирается *k* центров кластеров.
- Дальше в цикле делается следующее:
  - для каждой точки данных в наборе находится центроид с кратчайшим расстоянием;
  - когда все точки будут назначены центроиду, вычисляет сумму всех расстояний между элементом и его центроидом.
- Если расстояние оказалось не меньше, чем в предыдущей итерации, возвращаются кластеры.
- Каждый центроид перемещается в центр назначенного кластера.

Существует много способов выбрать начальные положения центроидов, и это очень важный шаг, от которого сильно зависят результаты всей кластеризации. В 10 главе книги Питера Харрингтона (Manning, 2012) «Machine Learning in Action» вы найдете методы, позволяющие сделать алгоритм кластеризации умнее. Я покажу здесь лишь краткий обзор того, как это работает, потому что понимание работы этих алгоритмов позволяет гораздо лучше понять, откуда взялся тот или иной результат, а также увидеть ошибки.

7.3.2. Реализация кластеризации k-средних на Python

Я покажу вам код, который поможет лучше понять алгоритм. Код не будет использоваться в приложении MovieGEEKs, потому что он подходит в качестве иллюстративного примера, но работает довольно медленно. Если вы хотите поиграть с этим кодом сами, откройте файл *Jupyter Notebook*, который можно найти в папке заметок проекта GitHub[1].

В начале нашего путешествия по кластеризации k-средних пойдем по легкому пути – выберем случайные две точки входных данных, которые будут служить в качестве начальных центров кластеров. Реализующий это код показан в следующем листинге.

ЛИСТИНГ 7.1. Генерация центроидов

```
import random

def generate_centroids(k, data):
    return random.sample(data, k)
```

Берет *k* случ йных элементов д нных

[1] Для получения дополнительной информации о Notebook Jupyter см. **mng.bz/KMfU**.

Для каждой точки данных в наборе ищем центроид с кратчайшим расстоянием до него

Следующее, что нужно сделать, – это рассчитать расстояние между элементами в данных и центроидах и найти что ближе. В вычислении расстояний нам поможет старый знакомый: евклидова норма, о которой мы говорили выше. Формула выглядит следующим образом:

$$r_{сара} - r_{петро2} = \sqrt[2]{\sum_{i=1}^{n} \left| r_{сара,\,i} - r_{петро,\,i} \right|^2}.$$

В коде эта формула реализуется следующим образом:

ЛИСТИНГ 7.2. Вычисление расстояния между двумя векторами

```
import math

def distance(x,y):
    dist = 0
    for i in range(len(x)):
        dist += math.pow((x[i] - y[i]), 2)
    return math.sqrt(dist)
```

Проход по всем измерениям

Доб вление кв д р т р зности между вектор ми в д нном измерении

Возвр щение кв др т суммы

Вы можете использовать метод `distance`, чтобы решить, к какому кластеру должен быть отнесен каждый элемент, как показано далее (листинг 7.3). Метод возвращает центроид с наименьшим расстоянием до элемента.

ЛИСТИНГ 7.3. Назначение элемента кластеру

```
def add_to_cluster(item, centroids):
    return item, min(range(len(centroids)),
                     key= lambda i: distance(item, centroids[i]))
```

Проверк всех центров кл стеров и возвр т ближ йшего

Когда все точки будут назначены центрам, метод `distance` вычислит сумму всех расстояний между элементом и его центроидом. Сумма используется для сравнения качества результатов итераций алгоритма. Во второй итерации результат сравнивается с суммой, полученной в прошлой итерации. Если предыдущий результат оказался лучше, то алгоритм останавливается, а в противном случае делает еще одну итерацию.

В каждой итерации центроиды перемещаются. Это можно сделать разными способами, но в данном листинге они перемещаются ближе к общему центру всех точек в кластере.

ЛИСТИНГ 7.4. Смещение центроидов в центр кластера

```
from functools import reduce

def move_centroids(k, kim):
    centroids = []
    for cen in range(k):
        members = [i[0] for i in kim if i[1] == cen]
        if members:
            centroid = [i/len(members) for i in reduce(add_vector, members)]
            centroids.append(centroid)

    return centroids
```

Проход *k* кл стеров для созд ния центроидов

Поиск всех членов этого кл стер (к ждый элемент – это кортеж из вектор и д нных прин длежности к кл стеру)

…скл дыв ет все векторы и делит н их количество. Получ ется точк в центре кл стер

Если кл стер не пустой…

Метод `add_vector` в листинге 7.5 работает довольно просто. Он перебирает векторы и складывает элементы, как показано в листинге ниже.

ЛИСТИНГ 7.5. Вспомогательный метод

```
def add_vector(i, j):
    return [i[k] + j[k] for k in range(len(j))]
```

Теперь давайте рассмотрим алгоритм кластеризации k-средних.

ЛИСТИНГ 7.6. Алгоритм кластеризации k-средних

```
def k_means(k, data):
    best_weight = math.inf

    centroids = generate_centroids(k, data)

    while True:
        iteration = list([add_to_cluster(item, centroids) for item in data])

        new_weight = 0

        for i in iteration:
            new_weight += distance(i[0], centroids[i[1]])

        if new_weight < best_weight:
            best_weight = new_weight
            new_weight = 0
        else:
            return iteratiasd+on

        centroids = move_centroids(k, iteration)

k_means(k, data)
```

Пок что лучший вес – бесконечность

Генер ция центроидов

Вычисление р сстояния между элемент ми и центроид ми

Если новый вес лучше прежнего – лгоритм продолж ется, если нет – прерыв ется

Перер счет центроидов

З пуск кл стериз ции

Нзн чение к ждой точки к кл стеру

Вычисление кластеров означает, что вы можете уменьшить количество пользователей, с которыми нужно сравнивать нового пользователя при вычислении сходства. На рис. 7.9 показано выполнение алгоритма в течение четырех итераций.

В результате мы получили три кластера, показанных в табл. 7.7. Для нашего детского примера вроде все в порядке.

Таблица 7.9. Примеры кластеров

Кластер	Элементы
0	Питер = [0, 6]
	Мела = [1, 4]
	Влад = [4,4]
	Ким = [5,3]
	Эгле = [2,1]
1	Хелла = [9,9]
	Джимми = [6,8]
2	Сара = [7,1]
	Дея = [10,0]

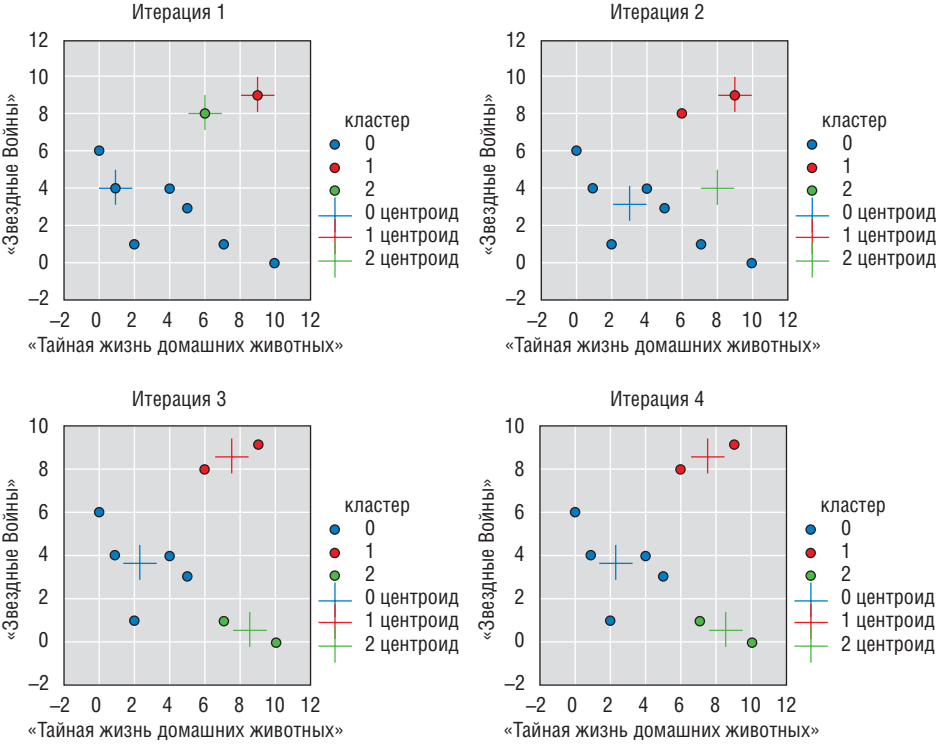


Рис. 7.9. Было выполнено четыре итерации кластеризации k-средних, прежде чем алгоритм прекратил работу

Выберем случайные значения и убедимся, что это будет работать. Например, к какому кластеру будет отнесен новый пользователь, которому понравилась «Тайная жизнь домашних животных», но «Звездные войны» не особо ([10, 4])?

Предупреждение

Прежде чем двигаться дальше, я думаю, будет справедливо сказать, что после показанного в табл. 7.7 примера (нормированные оценки Петро и Джаспера) легко поверить, что кластеризация k-средних – это волшебная палочка, исполняющая любые желания. Это не так. Кластеризация k-средних, как и большинство других алгоритмов машинного обучения, не будет идеально работать всегда и везде. Часто она дает такие результаты, которые нормально использовать не удастся.

Примеры в книгах всегда просты и призваны лишь проиллюстрировать и показать, как это работает. А вот когда что-то НЕ работает – это уже сложно объяснить, и читатель обычно выясняет все сам. На рис. 7.10 показаны другие результаты, которые я получил, запуская алгоритм из разных точек.

Боюсь, здесь я мало чем могу помочь далее. В следующем разделе вы увидите, как кластеризация k-средних реализована на сайте MovieGEEKs и выдает списки похожих пользователей. По крайней мере, это более реалистичный пример того, как этот алгоритм можно реализовать.

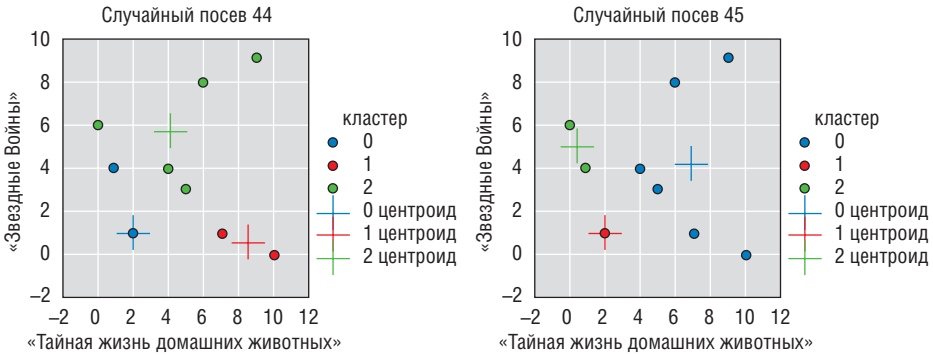


Рис. 7.10. Менее удачные результаты кластеризации k-средних

Как использовать кластеры

Вы можете использовать кластеры двумя способами:

1. *Просматривать пользователей в группах*. Если вам нужны пользователи, похожие, например, на Ким, достаточно найти ее кластер и вывести других пользователей в этой группе.
2. *Размещать новых пользователей в группы.* Что делать, когда появляется новый пользователь? Нужно будет найти центроид, находящийся ближе всего к новой точке данных. Благодаря этому можно определить, в какой кластер добавить нового пользователя.

Теперь вам наверняка не терпится реализовать кластеры, пока память еще свежа, но давайте вернемся к повествованию главы и добавим на сайте MovieGEEKs поиск сходства, а затем кластеры.

7.4. Реализация вычисления сходства

Имеет смысл рассматривать только пользователей, которые оценивали преимущественно те же фильмы, что и текущий пользователь. Если хочется быть еще разборчивее, то можно отбирать не только по фильмам, а еще и по качеству оценок. Именно этих пользователей можно вывести в список рекомендуемых.

Если вы посмотрите на множества, они будут выглядеть так, как изображено на рис. 7.11, и тут видно, какие множества имеют важное значение для совместной фильтрации пользователей.

Вкратце так: нам нужно найти пользователей, у которых много общего с активным пользователем (но не полное совпадение). Как можно это сделать? Если вы используете SQL, можно было бы сделать что-то вроде того, как показано в листинге 7.7. Мы ищем пользователей, похожих на пользователя с `id = 4`.

ЛИСТИНГ 7.7. Алгоритм кластеризации k-средних

```
WITH au_items AS (
SELECT
  distinct(movieid), rating
FROM
  public.ratings
WHERE
  userid = '4')

SELECT userid, count(movieid) overlapping
FROM public.ratings
WHERE movieid IN (SELECT movieid from au_items) and
userid <> '4'
AND overlapping > min
group by userid
order by overlapping desc;
```

Сбор элементов, которые оценил текущий пользов тель

Поиск всех пользов телей, которые оценили хотя бы один элемент из этих

Поиск оценок к фильм м, которые оценил текущий пользов тель

Группировк по идентифик тору

Порядок по перекрыв ющимся элемент м

Требуется перекрытие н более чем несколько минут

Пользов тели, отличные от текущего

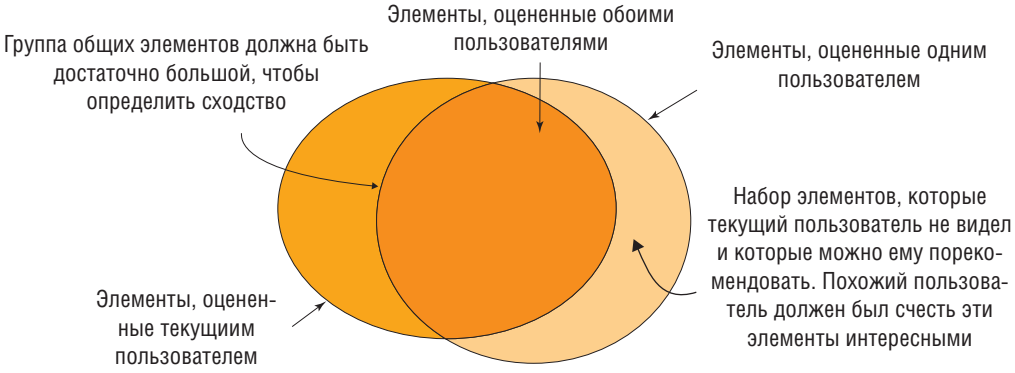


Рис. 7.11. Диаграмма показывает, какие наборы имеют важное значение для совместной фильтрации

Чтобы сделать такой запрос в Django, вы можете использовать `QuerySet`, и ваша процедура будет выглядеть так, как показано в листинге ниже. Запрос находится в папке */recommender/views/py* наGitHub.

ЛИСТИНГ 7.8. Отбор похожих пользователей с помощью Django ORM

```
ratings = Rating.objects.filter(user_id=user_id)
sim_users = Rating.objects.filter(movie_id__in=ratings.values('movie_id')) \
        .values('user_id') \
        .annotate(intersect=Count('user_id')).filter(intersect__gt=min)
```

Это медленный запрос, но если учесть, что вы убрали большой кусок пользователей, которые не схожи с текущим пользователем, то ожидание, возможно, стоит того. С таким списком достаточно посмотреть на 100 самых похожих пользователей или по меньшей мере схожих хоть в чем-то.

В поисках совета о том, как бы еще сократить список, я обнаружил, что Майкл Д. Экстранд и соавторы в GroupLens предлагают использовать около 20–50 пользователей[1]. У меня есть ощущение, что это многовато для большинства наборов данных, но для хорошей выборки вполне достаточно. Я призываю вас проверить это.

Далее мы рассмотрим, как реализовать вычисление коэффициента Пирсона, о котором мы говорили ранее. Метод, приведенный в листинге 7.9, сравнивает двух пользователей и вычисляет их сходство. Этот код тоже лежит в папке */recommender/views/py* наGitHub.

ЛИСТИНГ 7.9. Реализация вычисления коэффициента сходства Пирсона

```
def pearson(users, this_usr, that_usr):
  if this_usr in users and that_usr in users:
    this_sum = sum(users[this_usr].values())
    this_len = len(users[this_usr].values())
    this_usr_avg=this_sum/this_len
    this_keys = set(users[this_user].keys())
    that_keys = set(users[that_user].keys())

    all_movies = (this_keys & that_keys)
    dividend = 0
    divisor_a = 0
    divisor_b = 0

    for movie in all_movies:
        nr_a = users[this_user][movie] - this_user_avg
        nr_b = users[that_user][movie] - that_user_avg
        dividend += (nr_a) * (nr_b)
```

Поиск средних оценок пользов телей

Объединение двух множеств фильмов в один

Норм лиз ция оценок пользов телей путем вычит ния среднего

[1] GroupLens (**grouplens.org**), Исследовательская группа в университете Миннесоты, произвела много хороших исследований рекомендательных систем.

```
    divisor_a += pow(nr_a, 2)
    divisor_b += pow(nr_b, 2)

divisor = Decimal(sqrt(divisor_a) * sqrt(divisor_b))
if divisor != 0:
  return dividend/divisor

return 0
```

Вычисление коэффициент Пирсон

Возвр щ ет 0, если делитель нулевой

7.4.1. Реализация вычисления сходства на сайте MovieGEEKs

Чтобы облегчить вам понимание, мы посмотрим, как алгоритм вычисления сходства реализован в админке сайта MovieGEEKs. Вы заходили на MovieGEEKs (**mng.bz/04K5**)? Там есть аналитическая часть, которая, в свою очередь, содержит страницу для каждого user_id. Вы можете найти ее по адресу **http://localhost:8000/analytics/user/100s**.

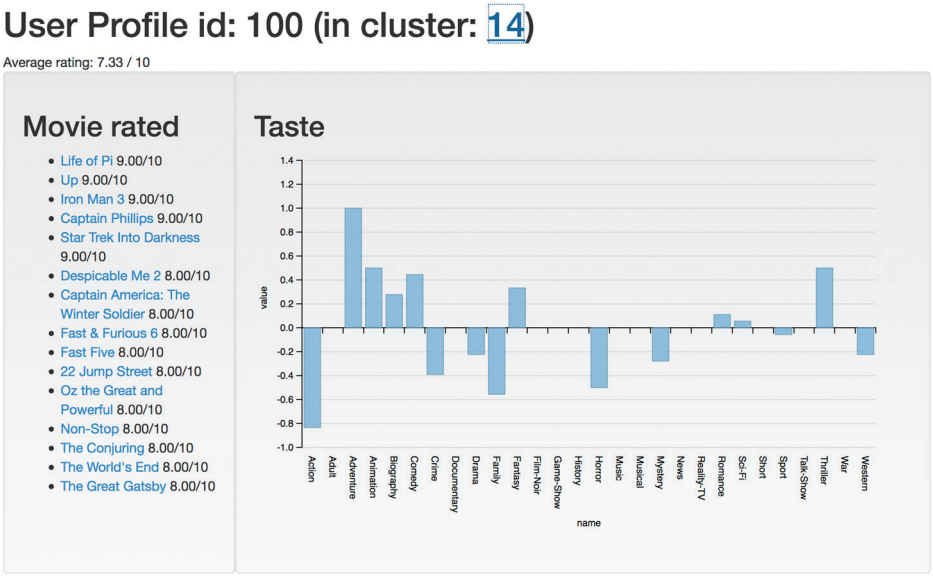


Рис. 7.12. Верхняя часть страницы профиля пользователя с идентификатором 100

Посмотрите на пользователя с идентификатором 100. Он любит много разных жанров, например приключения, мультики и триллеры, – это видно на графике из рис. 7.12 или на сайте.

Нам же требуется реализовать следующий раздел, в котором показываются похожие пользователи. Он изображен на рис. 7.13.

Рис. 7.13. Пользователи, похожие на пользователя 100, рассчитанные с помощью сходства Жаккара – слева, и коэффициента Пирсона – справа

На сайте MovieGEEKs расчет сходства является частью рекомендательной системы, так что я добавил метод `similar_users` к API рекомендатора, как показано в листинге 7.10. Для метода `similar_users` требуется метод `user_id` и тип. Тип позволяет легко расширить его другими видами расчетов подобия. Я оставлю его до заинтересованных читателей, а теперь перейдем непосредственно к коду метода на Python.

ЛИСТИНГ 7.10. similar_users: /recommnder/views.py

Сбор оценок текущего пользователя

Можно добавить минимальное перекрытие, если нужно

Сбор оценок всех пользователей, которые перекрываются с текущим

Извлечение всех пользователей, оценивших один или больше тех же фильмов

```
def similar_users(request, user_id, type):

    min = request.GET.get('min', 1)
    ratings = Rating.objects.filter(user_id=user_id)
    sim_users = \
      Rating.objects.filter(movie_id__in=ratings.values('movie_id'))\
                .values('user_id') \
                 .annotate(intersect=Count('user_id')) \
                 .filter(intersect__gt=min)
    users = {u['user_id']: {} for u in sim_users}
    dataset = Rating.objects.filter(user_id__in=users.keys())

    for row in dataset:
        if row.user_id in users.keys():
            users[row.user_id][row.movie_id] = row.rating

    similarity = dict()
    switcher = {
        'jaccard': jaccard,
        'pearson': pearson,
    }
```

Извлечение всех user_id

Построение матрицы пользователь-оценок

Хитрый способ реализовать оператор выбор на Python. Если нужно добавить другой метод подобия, можно добавить имя метод

```
for user in sim_users:
    func = switcher.get(type, lambda: "nothing")
    s = func(users, int(user_id), int(user['user_id']))
    if s > 0.2:
        similarity[user['user_id']] = s

data = {
    'user_id': user_id,
    'num_movies_rated': len(ratings),
    'type': type,
    'similarity': similarity,
}
return JsonResponse(data, safe=False)
```

Итер ция по всем пользов телям

Ссылк н функцию, опис нную н входе, и ее выполнение

Доб вление пользов теля в список похожих пользов телей

Проверк , превыш ет ли сходство 0,2. Изменения этого зн чения д дут большее или меньшее сходство

Возвр т JSON

Метод Пирсона подробно описан в листинге 7.9. Здесь используется метод `jaccard`. Более подробное описание приведено в разделе 7.2 ранее в этой главе. Вы найдете этот фрагмент кода в файле */recommender/views.py*.

ЛИСТИНГ 7.11. Метод Жаккара

```
def jaccard(users, this_user, that_user):
    if this_user in users and that_user in users:
      intersect = set(users[this_user].keys()) \
& set(users[that_user].keys())
union = set(users[this_user].keys()) |\
 set(users[that_user].keys())
return len(intersect)/Decimal(len(union))
  else:
return 0
```

Вычисление пересечения между двумя пользов телями

Вычисление объединения двух пользов телей

Возвр щение коэффициент корреляции Ж кк р

Я провел небольшой тест, чтобы посмотреть, одинаково ли работает коэффициент Пирсона и кластеризация k-средних. Я взял несколько пользователей, которых нашел в списке похожих для пользователя 2 со сходством 0,87 и 0,85 соответственно, и все они были в группе № 7. Не получится узнать, работает ли это во всех случаях. Однако этот пример, по крайней мере, дает хорошее представление о том, правильно ли алгоритм реализован в целом.

Вычисление сходства пользователей, которое мы рассмотрели, не требует какой-либо специальной подготовки. В главе 8 мы поговорим о сходстве элементов и рассчитаем все значения сходства заранее.

7.4.2. Реализация кластеризации на сайте MovieGEEKs

Использование алгоритма кластеризации на вашем сайте будет в чем-то сродни прыжку веры, но не беспокойтесь об этом. Здесь мы реализуем решение, используя алгоритм кластеризации, который является частью библиотеки Scikit-learn[1]. Это тоже алгоритм k-средних, но работает предположительно

[1] Scikit-learn – бесплатная библиотека для Python. См. **scikitlearn.org/stable/index.html**.

быстрее и лучше, чем в нашем маленьком примере, так что воспользуемся им. Мы добавим кластеры в аналитическую часть сайта MovieGEEKs (рис. 7.14) в двух местах и начнем с главной страницы **http://localhost:8000/analytics/**. Но мы пока не рассчитали кластеры, поэтому отображаться они не будут.

Сценарий, реализованный для сайта MovieGEEKs, делает то же самое, что и код в листинге 7.6, но теперь он загружает данные из базы данных, вычисляет k-средние, а затем сохраняет строку в таблицу Cluster для каждого `user_id` с соответствующим `cluster_id`, как показано в листинге 7.12. Вы найдете сценарий в файле */builder/user_cluster_calculator.py*.

ЛИСТИНГ 7.12. Сценарий UserClusterCalculator

```
class UserClusterCalculator(object):

def load_data(self):
    print('loading data')
    user_ids = list(
        Rating.objects.values('user_id')
            .annotate(movie_count=Count('movie_id'))
            .order_by('-movie_count'))
    content_ids = list(Rating.objects.values('movie_id').distinct())
    content_map = {content_ids[i]['movie_id']: i
                   for i in range(len(content_ids))}
    num_users = len(user_ids)
    user_ratings = dok_matrix((num_users,
 len(content_ids)),
dtype=np.float32)

for i in range(num_users):
       # each user corresponds to a row, in the order of all_user_names
       ratings = Rating.objects.filter(user_id=user_ids[i]['user_id'])
       for user_rating in ratings:
           id = user_rating,movie_id
           user_ratings[i, content_map[id]] = user_rating.rating
   print('data loaded')

   return user_ids, user_ratings

def calculate(self, k = 23):
    print("training k-means clustering")
    user_ids, user_ratings = self.load_data()
    kmeans = KMeans(n_clusters=k)
    clusters = kmeans.fit(user_ratings
```

Выбирает все user_id из таблицы оценок

Выбирает все content_id из таблицы оценок

Массив отображение между content_id и списком целых чисел, чтобы все работало с разреженной матрицей

Создание экземпляра матрицы слов ей ключей с размерностями[1], соответствующими пользователям и контенту

Проход по всем пользователям и добавление данных в матрицу

Создание экземпляра алгоритма кластеризации k-средних

Немного магии (кластеризация)

[1] См. **mng.bz/5bK4**.

```
.tocsr())

    plot(user_ratings.todense(), kmeans, k)

    self.save_clusters(clusters, user_ids)

    return clusters

def save_clusters(self, clusters, user_ids):
    print("saving clusters")
    Cluster.objects.all().delete()
    for i, cluster_label in enumerate(clusters.labels_):
        Cluster(
            cluster_id=cluster_label,
            user_id=user_ids[i]['user_id']).save()

if __name__ == '__main__':
    print("Calculating user clusters...")

    cluster = UserClusterCalculator()
    cluster.calculate(23)
```

Уд ление всех кл стеров, сохр ненных в б зе д нных, чтобы созд ть комн ту для новых

Сохр нение кл стеров

Этот сценарий можно запустить из командной строки (или непосредственно из PyCharm, которым я пользуюсь). Из командной строки можно просто написать код, приведенный в листинге ниже.

ЛИСТИНГ 7.13. Запуск кластеризации

```
python -m builder.user_cluster_calculator
```

На MacBook 2014 года процедура занимает несколько часов, так что вы можете сделать зарядку и перекусить, прежде чем мы продолжим.

Рис. 7.14. Страница загрузки сайта MovieGEEKs с панелью аналитики и список кластеров

Помимо вывода аналитики на загрузочной странице сайта MovieGEEKs, вы можете нажать один из кластеров (например, первый) и получите вид, показанный на рис. 7.15. Это диаграмма нормализованного распределения оценок всех членов кластера, а также список всех его членов.

Это глава призвана дать описание, обсудить ваше понимание и проверить, работает ли ваш алгоритм кластеризации так, как должен. Но использование кластеров может быть способом оптимизации алгоритмов, как описано в следующей главе, и вы можете использовать алгоритмы кластеризации, чтобы сузить диапазон пользователей и тем самым ускорить вычисление рекомендаций.

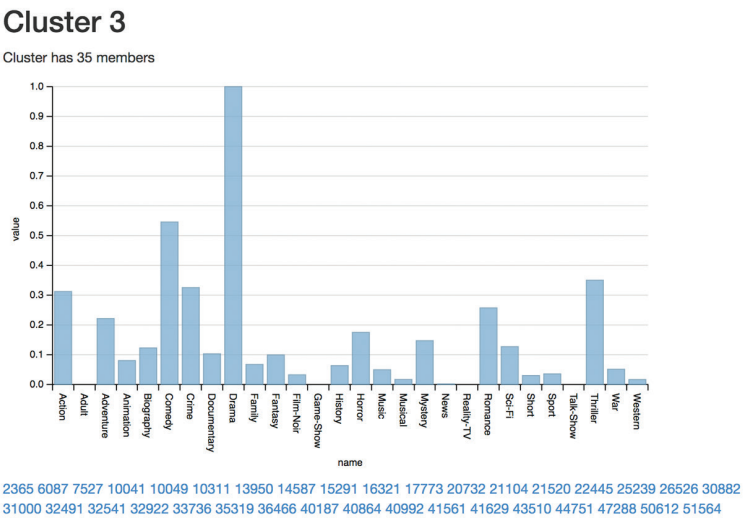


Рис. 7.15. Скриншот из аналитики MovieGEEKs с выводом кластера

Сходство и несходство в рекомендательных системах и во многих алгоритмах машинного обучения решают все. Дата-сайентисты часто проводят много времени, занимаясь преобразованием данных из категорий или текста в формат, который может быть использован для вычисления сходства, поэтому важно хорошо представлять в голове, как выглядят функции сходства.

В этой главе мы говорили о том, что данные, например, двумерных количественные данных могут быть визуализированы. Это здорово для демонстрации примеров, но в реальности обычно все не так. Я советую сначала хорошо ознакомиться с двумерными примерами сходства данных, которые мы привели здесь, потому что они помогут вам лучше понять более сложные ситуации в дальнешем.

Резюме

- Измерение сходства между множествами можно использовать для вычисления подобия пользователей, если имеются данные о покупках или лайках.
- Количественные данные представляют интерес, когда пользователи явно или неявно ставят оценки.
- L_p-нормы могут быть использованы на количественных данных, а также являются отправной точкой для использования других корреляционных методов, таких как коэффициенты корреляции Пирсона и Отиаи, которые выглядят похоже, но имеют различные интерпретации.
- Кластеризация k-средних отлично работает на игрушечных примерах, но результаты надо всегда проверять. Как вы видели, данные могут легко сгруппироваться неоптимально, что даст в итоге странные рекомендации для пользователей. (Мы привели пример в предупреждении и напомнили вам, что приведенные примеры лишь объясняют алгоритм, но утаивают тот факт, что многие алгоритмы трудно подогнать под свои потрености. Мы вас предупредили.)

Глава 8

Совместная фильтрация в окрестностях

Совместная работа всегда упрощает дело, поэтому давайте приступим:

- сначала мы вернемся к матрицам оценок;
- поговорим о теории совместной фильтрации;
- совместная фильтрация выполняется в несколько шагов, и мы рассмотрим каждый из них, а также узнаем, какие нужно будет принять решения;
- вы узнаете, как совместная фильтрация реализована на сайте MovieGEEKs.

В этой главе мы начнем изучать совместную фильтрацию и подробно поговорим о ее частном случае – *фильтрации в окрестности*. Совместная фильтрация включает в себя множество методов. Общей чертой для них всех является выбор данных. В этих методах фильтрации в качестве источника для создания рекомендаций используются только оценки (явные или неявные).

Я посвятил совместной фильтрации две главы – эту и главу 10. В главе 10 мы поговорим о моделях с использованием матричной факторизации, позволяющей найти скрытые функции. В главе 9 порассуждаем о фильтрации на основе контента[1].

В этой главе вы узнаете немного из истории рекомендательных систем и о различных способах использования совместной фильтрации. Основу совместной фильтрации, которую мы здесь рассмотрим, составляет фильтрация в окрестностях, основанная на сходствах между пользователями и элементами. А о расчетах сходства был разговор в главе 7.

До сих пор мы создавали простые неперсонализированные рекомендательные системы. Теперь пора поближе познакомиться с персональными рекомендациями. Созданные ранее рекомендации были основаны на автоматически

[1] Я также использовал совместную фильтрацию в главе 12, где рассматриваются гибридные рекомендаторы, а также в главе 13, где обсуждаются алгоритмы ранжирования. Но это там не главное.

генерируемых собираемых данных о поведении, но с этого момента мы будем брать реальный набор данных под названием MovieTweetings и построим на его основе рекомендации[1]. Я рекомендую также попробовать набор данных с GitHub, чтобы ознакомиться с ним.

Алгоритм совместной фильтрации работает просто. Есть всего несколько вещей, необходимых для выработки рекомендаций. На каждом из этапов создания рекомендаций есть несколько вариантов действий, которые влияют на результат. Рассмотрим каждый шаг подробно, чтобы понять все это.

После окончания этой главы вы будете знать, как реализовать совместный алгоритм фильтрации, используемый на Amazon. Ну, по крайней мере, именно такой они публиковали в 2003 году[2]. Я удивлен, что Amazon до сих пор не придумали ничего другого. Этот алгоритм используется для генерации страницы «Рекомендовано для вас» на Amazon. Мои рекомендации показаны на рис. 8.1. Как вы можете видеть, я купил книги по статистике и Джанго. Общая идея состоит в том, чтобы найти элементы, имеющие те же оценки, которые я оценил или купил.

Алгоритмы совместной фильтрации в окрестности были первыми алгоритмами, классифицированными как алгоритмы рекомендательных систем. Сперва немного истории.

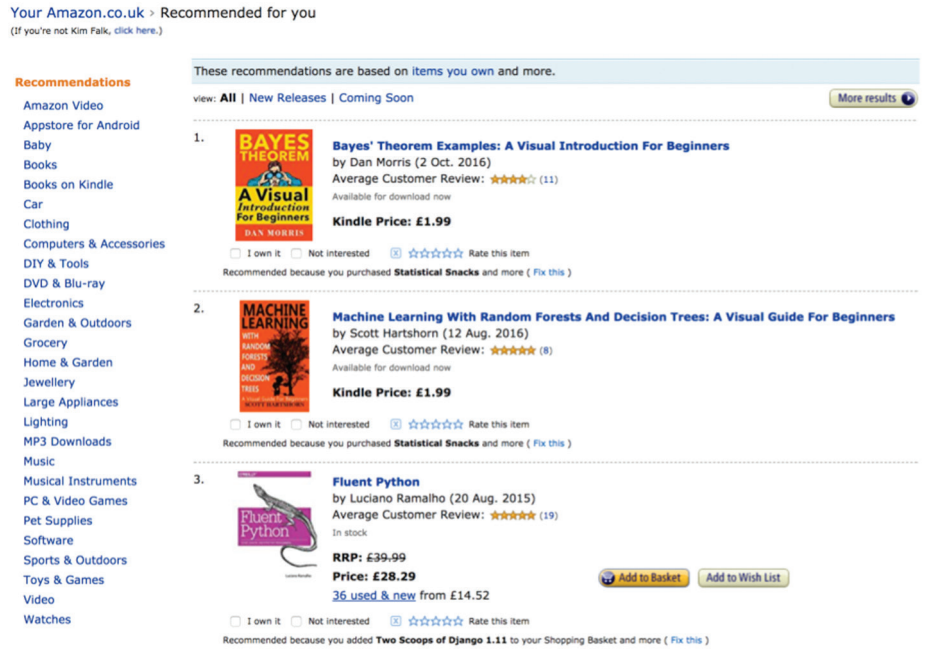


Рис. 8.1. Мои рекомендации на Amazon.com

[1] Дополнительные сведения о наборе данных MovieTweetings: **github.com/sidooms/MovieTweetings**.

[2] Г. Линден и др., «Рекомендации Amazon.com: Совместная фильтрация по элементам». Доступно через интернет по адресу **ieeexplore.ieee.org/abstract/document/1167344/**.

8.1. Совместная фильтрация: историческая справка

Большинство людей считают себя уникальными и не любят быть классифицированы. Но именно в этом и заключается суть совместной фильтрации для расчета рекомендаций. Простыми словами, совместная фильтрация выводит вам список рекомендуемых элементов. Список создается на основе данных о людях, чьи вкусы схожи с вашими и которые видели что-то, чего еще не видели вы.

8.1.1. Когда начали использовать совместную фильтрацию

Наша история началась в 1992 году в Xerox PARC (Palo Alto Research Center), когда они поняли, что количество отправляемых сообщений выросло до невероятных размеров, «… в результате чего пользователей просто завалило огромными потоками входящих документов»[1]. Уверен, что в 1992 году они еще ничего не знали, но, как и во многих других случаях, Xerox PARC опережали свое время, и информационная перегрузка, видимо, тоже пример этого.

Их почтовая система была основана на предположении, что всегда существует несколько пользователей, которые читают все и сразу, а затем отвечают на то, что им интересно, в то время как большинство пользователей будет читать только интересные им вещи. Почтовая система назывались Tapestry, и это название часто фигурирует в литературе по рекомендательным системам.

Два года спустя проект GroupLens, совместное предприятие МТИ и Университета Миннесоты, создал «открытую архитектуру для совместной фильтрации новостей»[2]. Проект GroupLens (**grouplens.org**) был призван решить ту же самую проблему информационной перегрузки. Идея была в том, чтобы информационные сообщения можно было бы оценить. На этот раз система была построена на предположении о том, что люди, которые ранее согласились с оценками, вероятно, согласятся с ними снова.

Xerox и GroupLens заложили основу для большей части из того, что мы теперь видим в рекомендательных системах. В следующем разделе мы поговорим о том, что сделали в GroupLens, а также об улучшениях, которые были внедрены с тех пор.

8.1.2. Взаимопомощь

Предположение, на котором базируется совместная фильтрация, заключается в том, что «вместе работается эффективнее». Звучит красиво и забавно, как финал эпического голливудского блокбастера, но что поделать – такова

[1] Д. Голдберг и др., «Использование совместной фильтрации в системе Tapestry» (1992). Доступно онлайн: **citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.104.3739**.

[2] Более подробная информация: **ccs.mit.edu/papers/CCSWP165.html**.

идея. Кроме того, здесь предполагается, что люди мало меняют свои вкусы с течением времени, и, если вы в чем-то с кем-то согласились в прошлом, вы, вероятно, согласитесь с ним и в будущем. Давайте добавим больше конкретики, прежде чем мы углубимся в теорию совместной фильтрации и перейдем к расчетам.

В главе 6 мы рассмотрели рекомендации, которые были основаны на том, что люди купили в прошлом. Теперь основное внимание будет уделяться пользователям. Вопрос стоит примерно так: «Если бы пользователь был корзиной товаров, что бы в ней лежало?»

В книжных магазинах и библиотеках часто можно встретить плакат с чем-то вроде этого: «Если вам понравилась эта популярная книга *X*, вам стоит попробовать эту (возможно, не столь популярную) книгу *Y*». Такие плакаты адресованы большой группе людей и нередко работают хорошо. Но нам нужно создавать рекомендации не общие, а индивидуальные или хотя бы нацеленные на небольшие группы схожих пользователей. Печатать и вешать на стену ничего не нужно, а нужно лишь вывести их на страницу, когда пользователь заходит на ваш сайт.

Фильтрация в окрестности может быть реализована двумя способами. Вы можете найти пользователей, чьи вкусы схожи с вашими, а затем рекомендовать к просмотру то, что они уже оценили, а вы еще нет, – это фильтрация по пользователям (на рис. 8.2 посмотрите в верхний левый угол, а затем вправо и вниз). Второй путь – найти элементы, похожие на другие элементы, которые вам понравились (начните в верхнем левом углу, а затем двигайтесь вниз и вправо по рис. 8.2), – это фильтрация по элементам. Сходство между пользователями и элементами рассчитывается на основе оценок.

Для первого способа нам нужно вычислить схожих пользователей. Мой друг Томас и я знаем друг друга много лет, и я абсолютно уверен, что ему нравятся те же фильмы, что и мне. Если я хочу пойти в кино, я могу спросить у него, что бы посмотреть. Если у меня много таких друзей, я мог бы спросить каждого из них, а затем использовать все их ответы. Группа будет совместно фильтровать имеющиеся фильмы и выдаст мне рекомендации.

Предположим, что вы посмотрели новый «Стартрек» и теперь хотите что-то подобное. Вы спросите всех своих друзей, что они думают о «Стартреке» и могут ли они порекомендовать что-то еще столь же классное. Я спросил Хеллу, которая, насколько мне известно, любит фантастику, и она порекомендовала мне «Изгой-один: Звездные войны. Истории». А вот Петро не питает теплых чувств к фантастике и говорит мне, что никогда не стал бы смотреть «Изгой-один».

Сложив одно и другое, я могу сделать вывод, что эти два фильма похожи (одному он понравился, а другому – нет). Поскольку «Стартрек» мне понравился, наверняка «Изгой-один» мне тоже придется по душе. Таким образом, друзья совместно помогли мне выбрать следующий фильм, на который я могу пойти посмотреть в кино (если не потрачу все свободное время на эту книгу).

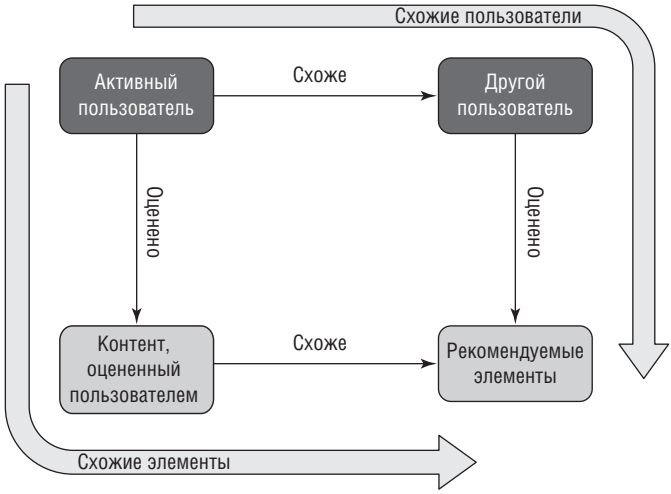


Рис. 8.2. Два способа фильтрации в окрестностях. В одном способе используются схожие пользователи, а в другом – элементы, схожие с теми, которые пользователю понравились

То есть мы находим пользователей, схожих с активным пользователем, а затем рекомендуем ему фильмы, которые понравились схожим пользователям. И второй способ – мы находим элементы, схожие с теми, которые пользователю понравились ранее. Чтобы сделать все это, нам снова понадобятся матрицы оценок, где описаны предпочтения пользователей.

8.1.3. Матрица оценок

Чтобы представить вкусы, нам нужно перечислить весь контент, который они оценили. Как правило, эти данные хранятся в матрице пользователь–элемент, которыми мы пользовались в 7 главе.

Матрица – это придуманное злыми математиками слово для обозначения таблицы с числами (пример показан в табл. 8.1). В каждой ячейке указано отношение пользователя к элементу. Как правило, оценка 5 или 10 является хорошей (в зависимости от того, какая используется шкала), а близкие к нулю оценки говорят о плохом отношении.

Вам нужно вычислить прогноз для каждой из пустых ячеек в этой матрице – это оценки, которые, вероятно, пользователь поставил бы конкретному контенту, основываясь на данных, имеющихся в матрице. Означает ли это, что нам требуется наличие всех данных? Не совсем, потому что в этом случае получается, что пользователь уже оценил весь предложенный контент, что, с одной стороны, хорошо, но тогда ему пора идти на другой сайт. Если матрица слишком пуста, то у вас возникает проблема холодного старта, о которой мы говорили в главе 6.

Таблица 8.1. Пример матрицы оценок. Обратите внимание на то, что Сара не оценила фильм «Эйс Вентура», а Джаспер и Хелла не оценили «Храброе сердце»

						
	Комедия	**Экшн**	**Комедия**	**Экшн**	**Драма**	**Драма**
Сара	5	3		2	2	2
Джаспер	4	3	4		3	3
Тереза	5	2	5	2	1	1
Хелла	3	5	3		1	1
Петро	3	3	3	2	4	5
Екатерина	2	3	2	3	5	5

Ну и как это связано с тем, что я сказал о выявлении схожих пользователей и элементов? Если вы посмотрите на таблицу, вы увидите, что у Сары и Терезы схожие вкусы, поэтому Сара может порекомендовать Терезе что-то новое, что ей понравится. Саре понравился фильм «Люди в черном», значит, можно узнать, какие еще фильмы ей по душе. На примере «Эйса Вентуры» видно, что некоторым он понравился (Джаспер и Тереза), а некоторым – нет (Хелла, Петро, и Екатерина), так же, как и «Люди в черном». Значит, эти фильмы похожи. Но это простой пример, и надо помнить, что пустых клеток обычно больше, чем заполненных, а для совместной фильтраци требуется чуть больше. Как решить эту проблему?

8.1.4. Процедура совместной фильтрации

Когда речь идет о машинном обучении и построении прогнозов, обычно имеется в виду последовательность вычислений, выполняемых в определенном порядке, после чего могут быть сделаны прогнозы. На рис. 8.3 приведен пример такой схемы.

В главе 7 вы узнали, что существует несколько способов вычислить сходство. В шаге 1 на рис. 8.3 показан один из способов. Позже вы узнаете о том, какие функции нужно использовать. Взяв один из способов расчета сходства, описанных в главе 7, вы можете понять, похож ли активный пользователь на других. На шаге 2 другие пользователи выстроены по порядку. На шаге 3 выбираются окрестности для расчета прогнозов. Опять же, существует много способов определить окрестности, но сейчас просто представим, что это близкая группа пользователей, похожих на активного пользователя. Позже в этом разделе мы рассмотрим способы, как это можно сделать.

На шаге 4 мы используем сходство пользователей в окрестности и оценки этих пользователей для элементов, чтобы составить прогноз. Здесь предска-

занная оценка равна 3,48. Прогнозируемые оценки можно использовать как есть или можно рассчитать побольше оценок и вывести, например, топ-*N* рекомендаций по этим оценкам.

Цель большинства систем заключается в том, чтобы сделать как можно больше работы, до того как жаждущий рекомендаций пользователь зайдет на сайт. Давайте посмотрим на то, что вы можете сделать.

8.1.5. Нужно использовать совместную фильтрацию пользователь–пользователь или элемент–элемент?

Какой тип совместной фильтраци нужно взять? Начнем с первого алгоритма, где мы ищем подобных пользователей и используем их для расчета рекомендаций. Средний пользователь ставит мало оценок, что означает, что добавление новой оценки может изменить рекомендации.

Считается неразумным делать предварительный расчет схожих пользователей. Элементы считаются более устойчивыми, поскольку одним и тем же типам людей нравятся одни и те же типы элементов, и исследования показали, что сходство элементов можно рассчитать заранее. Это важно, когда речь идет о каталоге размером с Amazon.

Еще разок посмотрите на рис. 8.2. Предположим, что вы хотите вычислить как можно меньше сходства. Если пользователей у вас намного больше, нужно делать фильтрацию по элементам. В противном случае выгоднее будет фильтровать по пользователям.

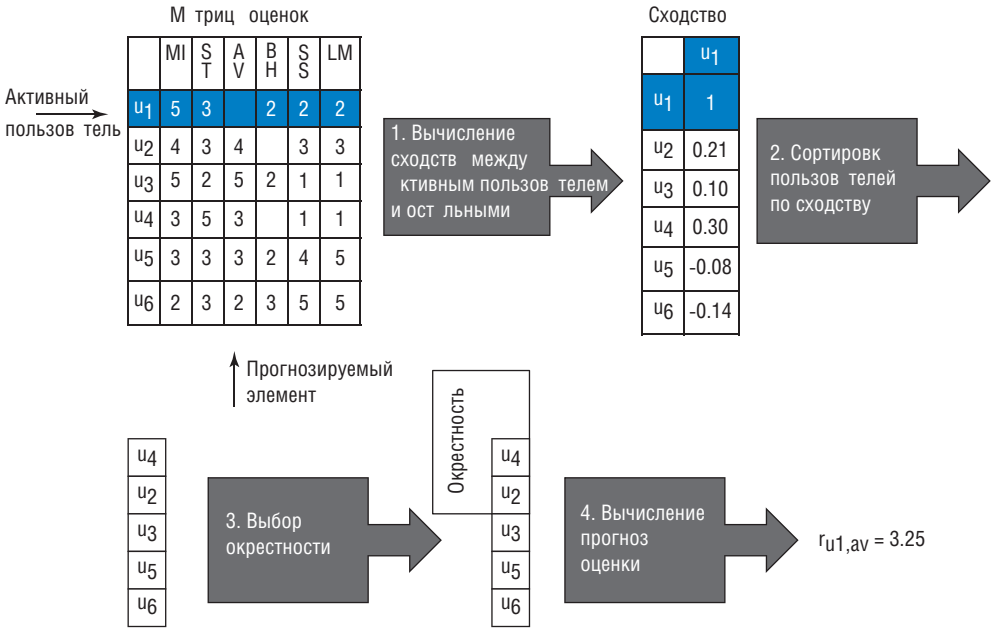


Рис. 8.3. Порядок фильтрации в окрестности по пользователям

В своем стремлении к знаниям вы можете встретить термин *фильтрация пользователь–пользователь*. Люди до сих пор говорят о такой фильтрации, так как это лучший способ формировать рекомендации. Если вы реализуете фильтрацию элемент–элемент, вы найдете элементы, похожие на те, которые пользователь А уже оценил, но подобные элементы не дают той информации, какую дают подобные оценки.

В случае со сходством пользователей мы надеемся на то, что ваши данные будут соединять пользователя с другими пользователями с различными особенностями вкусов, и мы получим хорошие рекомендации. Если вы хотите объяснить, почему дали эти рекомендации, то фильтрация по элементам облегчает эту задачу. Все потому, что система может сказать, что вам рекомендован фильм *Y*, так как вам понравился фильм *X*, который похож на *Y*. А в случае фильтрации по пользователям гораздо труднее объяснить, откуда взялись именно такие рекомендации.

8.1.6. Требования к данным

Соответствуют ли данные потребностям совместной фильтрации? Я много раз замечал, что нет четких правил о том, какой именно рекомендательный алгоритм следует использовать. Но, несмотря на это, я дам вам совет, прежде чем мы отправимся в захватывающий мир совместнй фильтрации. Для расчета рекомендаций данные должны быть хорошо связаны:

- если ни один пользователь не оценил контент, рекомендаций даваться не будет;
- пользователи, у которых нет перекрывающихся вкусов с другими пользователями, не будут получать хорошие рекомендации.

Для осуществления совместной фильтрации нужно сперва выявить все элементы, которые были оценены несколькими пользователями (хотя бы более двух). Затем мы считаем количество пользователей, соединенных с одним или несколькими из этих элементов. Эти пользователи будут получать рекомендации, а остальные не будут.

В совместной фильтрации хорошо то, что вашей системе не требуются какие-либо знания о предметной области. А вот вам как раз нужны специальные знания, чтобы создать хорошую рекомендательную систему.

8.2. Расчет рекомендации

Мы рассмотрели фильтрацию по пользователям и элементам. Мы и дальше будем говорить о фильтрации по пользователям, но, вероятно, в конечном итоге придем к расчетам сходства элементов, особенно если у вас, например, около 45 000 пользователей и всего 25 000 элементов. Порядок действий в этом случае чуть-чуть отличается – мы делаем то же, что и на рис. 8.3, но смотрим уже на элементы, а не на пользователей. На рис. 8.4 показаны действия, которые выполняются при фильтрации по элементам.

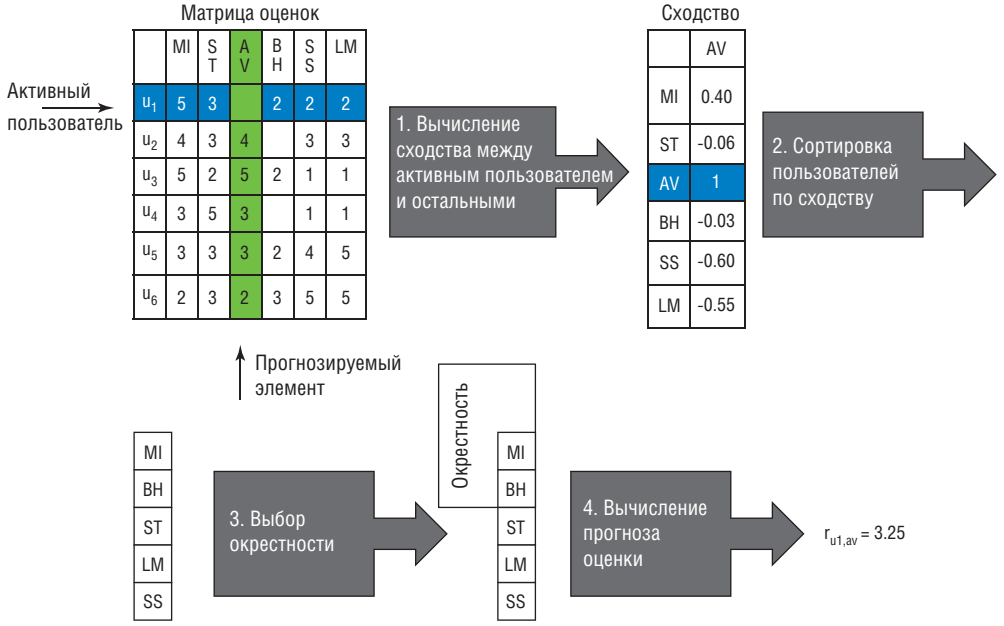


Рис 8.4. Процедура фильтрации по элементам

8.3. Расчет сходства

Первое, что вам нужно сделать, – это определиться с функцией сходства, о чем мы говорили в главе 7. Тут мнения расходятся. Мы возьмем коэффициент Отиаи, рассмотренный в главе 7. Он дает нам матрицу сходства, которая предоставляет перечень похожих элементов для каждого элемента.

8.4. Алгоритм вычисления сходства элементов с Amazon

Предполагается, что сходство элементов устойчиво, поэтому его можно вычислить предварительно или в офлайн. Amazon был одним из первых, и, вероятно, самым большим пользователем этого алгоритма, и они опубликовали статью, где описан их метод[1]. Грег Линденс, создатель большей части ранней системы рекомендации компании Amazon.com, описал псевдоалгоритм совместной фильтрации по элементам так:

```
Для каждого товара в каталоге, T1
    Для каждого клиента K, который купил T1
        Для каждого товара T2, приобретенного клиентом K
            Записать, что клиент приобрел T1 и T2
```

[1] Линден и др, «Рекомендации Amazon.com: Совместная фильтрация элемент–элемент». Доступно по ссылке **www.cs.umd.edu/~ Samir/498/Amazon-Recommendations.pdf**.

```
Для каждого товара T2
    Вычислить сходство между T1 и T2
```

Используя этот алгоритм, вы в конечном итоге получите набор данных, в котором можно найти элементы, подобные текущему элементу, что позволяет быстрее вычислять предсказания. В статье Amazon есть ссылка на статью Б. М. Сарвара и др.[1], которая также является отличным источником знания по совместной фильтрации элемент–элемент.

Вернемся к нашему примеру. Таблица 8.2 аналогична табл. 8.1, просто повторим ее тут для удобства.

Таблица 8.2. «Эйс Вентура»

						
	Комедия	**Экшн**	**Комедия**	**Экшн**	**Драма**	**Драма**
Сара	5	3		2	2	2
Джаспер	4	3	4		3	3
Тереза	5	2	5	2	1	1
Хелла	3	5	3		1	1
Петро	3	3	3	2	4	5
Екатерина	2	3	2	3	5	5

Все клиенты оценили «Людей в черном», так что теперь нужно посмотреть на каждого пользователя в отдельности. Первый пользователь – Сара, которая также оценила «Стартрек» (СТ), «Храброе сердце» (Х), «Разум и чувства» (РЧ) и «Отверженных» (О). Вам нужно добавить ее оценки в список оцененных элементов вместе с «Людьми в черном», поэтому у вас есть: ЛВЧ: [СТ, Х, РЧ, О]. Далее идет Джаспер, которые оценил все те же фильмы и «Эйса Вентуру» (ЭВ). Элемент должен быть добавлен только один раз, так что теперь у вас получается следующий список: ЛВЧ: [СТ, Х, РЧ, О, ЭВ].

И так далее-далее-далее для всех пользователей, и мы получим вот такой результат:

- ЛВЧ: [СТ, Х, РЧ, О, ЭВ]
- СТ: [ЛВЧ, Х, РЧ, О, ЭВ]
- ХС: [ЛВЧ, СТ, РЧ, О, ЭВ]

[1] Б. М. Сарвар и др., «Рекомендательные алгоритмы совместной фильтрации по элементам». X Международная Конференция по Всемирной Паутине (ACM Press, 2001), стр. 285-295.

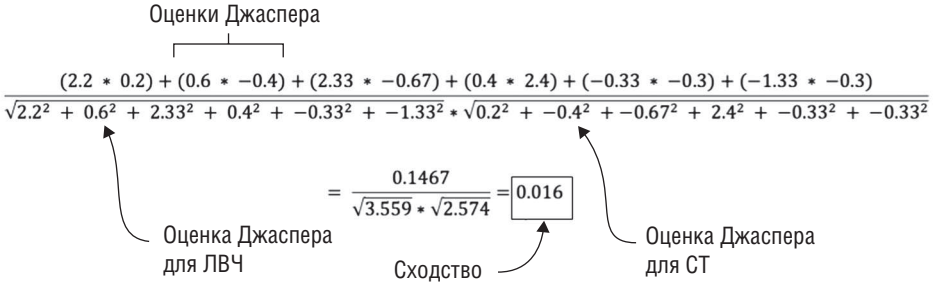
- РЧ: [ЛВЧ, СТ Х, О, ЭВ]
- О: [ЛВЧ, СТ, РЧ, Х, ЭВ]
- ЭВ: [ЛВЧ, СТ, Х, РЧ, О]

С помощью этих списков вы вычисляете сходства каждого из элементов в списке с ЛВЧ. Опять же, я покажу пример для первого списка, а вы сами сможете посчитать остальные. Мы будем использовать скорректированную функцию сходства Отиаи (где оценки нормализуются по средним оценкам для пользователя). Для расчета примем, что $nr_{i,u} = r_{i,u} - \bar{r}_u$:

$$sim(\text{"ЛВЧ","СТ"}) = \frac{\sum_u nr_{ЛВЧ,u} nr_{СТ,u}}{\sqrt{\sum_u nr^2_{ЛВЧ,u}} \sqrt{\sum_u nr^2_{СТ,u}}}.$$

Вынужден признать, что у меня проблема. Я пытался выписать результаты, но либо пример настолько мал, что результат был слишком очевиден, либо я сделал что-то не так. Делайте расчеты аккуратно, и все получится.

Сперва нужно нормализовать табл. 8.2, и результат приведем в табл. 8.3. Опять же, это делается путем вычисления средней оценки пользователя и вычитания этого числа из оценок. (тут другого пути нет, но в коде вы будете использовать кое-что посложнее). Имея нормализованные оценки, вы можете вычислить сходство между ЛВЧ и СТ. Чтобы сделать это, смотрите на каждого пользователя и суммируйте. Перемножайте каждую пару оценок для каждого пользователя и складывайте их (я отметил вклад Джаспера в уравнение):



Вы можете подумать: «Хмм… и что мне с этого?» Вы знаете, что скорректированная функция сходства Отиаи возвращает результат в пределах между –1 и 1. Это можно интерпретировать оценку выше среднего или ниже среднего от большинства людей.

Если вы посмотрите на цифры СТ и ЛВЧ в табл. 8.3, вы увидите, что Сара и Хелла дали фильму средние оценки, а Петро и Екатерина дали оценки ниже среднего. Джасперу и Терезе один фильм понравился больше, чем в среднем, а другой не понравился. Джаспер и Тереза вносят сумятицу, но Сара и Хелла дали разные оценки, так что никто из пользователей не смог сойтись во мнениях о двух фильмах. Это видно по значению сходства 0,016.

Таблица 8.3. Повторение табл. 8.2 с нормированными оценками по средней оценке пользователя

						
	Комедия	**Экшн**	**Комедия**	**Экшн**	**Драма**	**Драма**
Сара	2,20	0,20		−0,80	−0,80	−0,80
Джаспер	0,60	−0,40	0,60		−0,40	−0,40
Тереза	2,33	−0,67	2,33	−0,67	−1,67	−1,67
Хелла	0,40	2,40	0,40		−1,60	−1,60
Петро	−0,33	−0,33	−0,33	−1,33	0,67	1,67
Екатерина	−1,33	−0,33	−1,33	−0,33	1,67	1,67

Если сходство близко к 1, то мнения пользователей сошлись (неважно, в плохом или хорошем смысле). Если все пользователи положительно оценили один и отрицательно другой, сходство близко к –1. Если корреляции между оценками пользователей нет, сходство близко к 0. Я подсчитал все сходства в табл. 8.4.

Таблица 8.4. Матрица сходства между шестью фильмами. Отрицательные значения выделены темнее, а положительные – на белом фоне

						
ЛВЧ	1	0,63	1	−0,21	−0,88	−0,83
СТ	0,63	1	0,35	−0,47	−0,64	−0,62
ЭВ	1	0,35	1	0,01	−0,89	−0,83
Х	−0,21	−0,47	0,01	1	−0,23	−0,32
РЧ	−0,88	−0,64	−0,89	−0,23	1	0,96
О	−0,83	−0,62	−0,83	−0,32	0,96	1

Для того чтобы получить представление о том, как распределяется сходство, можно посмотреть на значения, близкие к 1, –1 или 0:

- близко к 1: О и РЧ – интересная пара, так как они отличаются от всех, кроме самих себя. У них высокое сходство, так как все пользователи сошлись во мнениях. Но тогда почему сходство не равно 1? Это потому что

оценки не в точности одинаковые, иначе сходство равнялось бы 1, как у ЭВ и ЛВЧ;

- близко к –1: РЧ и ЭВ максимально непохожи, что также понятно. Я могу представить себе, что большинству людей один из этих фильмов понравится, а другой – нет;
- близко к 0: У «Эйса Вентуры» и «Храброго сердца» сходство близко к нулю. Это указывает на то, что кому-то «Эйс Вентура» понравился, кому-то нет, и так же с «Храбрым сердцем». В этом примере только три пользователя оценили оба фильма, так что информации маловато. Два пользователя оценили оба фильма ниже среднего, а одному понравился «Эйс Вентура», но не понравилось «Храброе сердце».

Когда я считал это на Python, я мог бы добавить условие и выводить сходство только при значениях выше нуля. Это сделало бы предыдущие списки намного короче и проще для восприятия. Но в некоторых случаях может быть также целесообразно использовать отрицательные черты (то бишь несходство). Можно ли рекомендовать «Эйса Вентуру» людям, которым не понравился фильм «Разум и чувства»?

Если проблема повторяется – берегись!

В предыдущем алгоритме, если у элемента была только одна оценка, средняя оценка равнялась ей. Это означает, что функция сходства будет неопределенной.

Далее, если у вас есть один-единственный пользователь, который оценил некоторые два элемента, то сходство будет равно 1, что является самым высоким значением.

Будьте осторожны при вычислении сходства между пользователями, у которых слишком мало общих элементов. Представьте себе, что только Хелен посмотрела ЛВЧ и СТ и оценила их по-разному, но формула даст результат 100 % сходства:

$$\frac{2,17 \times 1,7}{\sqrt{2,17^2}\sqrt{1,7^2}} = 1.$$

Всегда требуется какое-то количество перекрывающихся пользователей – т. е. тех, кто оценил оба фильма. Не следует использовать совместную фильтрацию на пользователях, которые оценили только один или два элемента. Если перекрывающихся пользователей слишком много, то также может оказаться трудно найти сходство, поскольку рекомендации получатся слишком общими. Если всем понравилось несколько предметов, это превращается в график.

> **ПРИМЕЧАНИЕ.** Помните, что это компромисс. Слишком уменьшив количество пользователей, вы рискуете не найти рекомендаций вообще. Слишком увеличив – получите рекомендации, которые окажутся не по вкусу пользователю.

На рис. 8.6 показано, сколько пользователей оценило всего один элемент оценки. Почти 20 000 из 45 000 пользователей оценило только один фильм в наборе данных MovieTweetings.

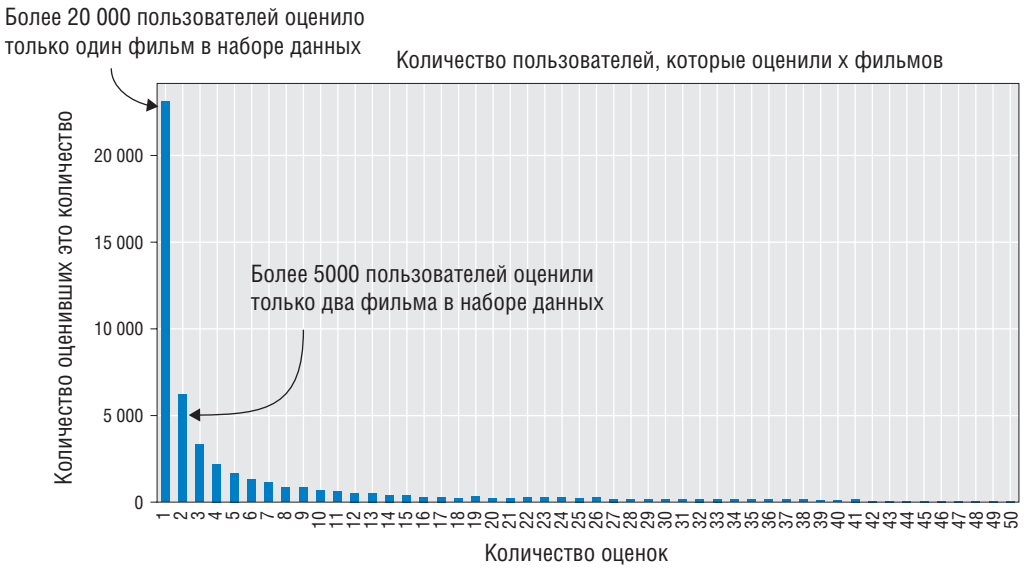


Рис. 8.6. Гистограмма показывает, сколько пользователей оценили определенное количество фильмов. Например, чуть больше 5000 оценили два фильма (при x = 2)

8.5. Способы выбора окрестности

Окрестность – это просто набор элементов, похожих на контент, который вы сейчас смотрите. Между этими элементами несходство (или расстояние) мало. В данном разделе я расскажу о трех способах определения и вычисления таких окрестностей: кластеризация, топ-*N* и пороговое значение.

Кластеризация

В главе 7 вы узнали, как реализовать кластеризацию. Если вы реализуете фильтрацию по пользователям, можно использовать эти кластеры в качестве окрестностей, либо можно отдельно выполнить кластеризацию элементов и использовать ее фильтрации по элементам. Чтобы выполнить кластеризацию с этой целью, вы, вероятно, захотите сделать побольше кластеров, чтобы полученные окрестности были не слишком велики, но суть останется такой же. Проблема этого метода заключается в том, что активный пользователь может оказаться на границе кластера или что у кластера будет странная форма, и окрестность получится неудачной.

Вместо использования кластеров в качестве окрестностей непосредственно вы можете применить кластеры для оптимизации алгоритма. Это сужает область, в которой система будет искать окрестность. Если вы разбили элементы на два кластера, возможность изучать только один из них даст большой скачок производительности.

> **ПРИМЕЧАНИЕ.** Сужая пространство, где вы ищете окрестность, вы также рискуете получить неоптимальный результат, так что будьте осторожны.

Я предлагаю делать тестовые прогоны с кластеризацией и без нее и сравнивать результаты с точки зрения качества (в главе 9 мы поговорим о том, как оценить рекомендательную систему). Если вы применяете кластеризацию, чтобы сузить область поиска, вы можете использовать ее вместе с методами топ-*N* или пороговым значением.

Топ-*N*

Самый простой способ найти окрестность – это определить число *N* как число соседей, которые должны находиться в окрестности, а затем сказать, что у всех элементов есть *N* соседей. Это позволяет системе всегда иметь элементы, с которыми можно работать, но некоторые из них могут оказаться не подобными. Это показано на рис. 8.7 с правой стороны.

В примерах на рисунках даны результаты при выборе $N = 3$ соседей. В первом примере нашлись точки, близкие к активной точке, а во втором искать пришлось подальше. Метод топ-*N* может заставить систему использовать точки, далекие от активной, что может дать плохие рекомендации. Для обеспечения лучшего качества можно использовать описанный далее подход.

Пороговое значение

Еще один способ нарезать наш торт из фильмов – это сказать, что в окрестности могут быть только элементы, имеющие сходство не ниже заданного. Это показано на левой стороне рис. 8.7.

У первой точки на рисунке это работает прекрасно, и подобные элементы получились такими же, как у метода топ-*N*. А вот со второй точкой есть проблемы, потому что у нее нет каких-либо близлежащих точек. Окрестность становится маленькой и может даже оказаться пустой.

Рис. 8.7. Два способа нахождения окрестностей. Слева показано, как работает метод порогового значения. Вокруг активной точки рисуется круг (по крайней мере, в 2D), и все элементы внутри него будут являться соседями. Справа показан метод топ-*N*. Здесь уже соседи выбираются не по расстоянию, а набираются до нужного количества

> **СОВЕТ.** Выбор между методами топ-*N* и пороговым значением – выбор между количеством и качеством. Пороговое значение – это качество, а топ-*N* – количество.

Независимо от выбранного метода встает второй вопрос: каким должно быть значение порога или число *N*? В методе выбора окрестности топ-*N* вам нужно найти постоянное значение количества соседей *N*. И пороговое значение тоже надо выбрать постоянным. Выбор зависит от ваших данных и от того, насколько качественные рекомендации вы хотите получить.

В идеальном наборе данных все элементы будут равномерно распределены по всей площади, и выбрать число *N* будет легко. Но это случается не часто, и это вопрос баланса между качеством рекомендаций для людей, любящих популярные вещи, или для людей со своеобразными вкусами.

8.6. Поиск правильной окрестности

Возвращаясь к примеру, описанному в разделе 8.4, давайте теперь посмотрим, что произойдет, если мы используем метод топ-*N* или пороговое значение. Возьмем значения сходства, вычисленные в разделе 8.4, и сведем в табл. 8.5 полученные окрестности.

Таблица 8.5. Результаты расчетов окрестности. Пороговое значение возвращает подобные элементы, а топ-2 дает не слишком схожие результаты

Фильмы	Топ-2	Порог: 0,5
«Люди в черном» (ЛВЧ), «Стартрек» (СТ) «Эйс Вентура» (ЭВ)	СТ: 0,63, ЭВ: 1,00 ЛВЧ: 0,63, ЭВ: 0,35 ЛВЧ: 1,00, СТ: 0,35	СТ: 0,63, ЭВ: 1,00 ЛВЧ: 0,63 ЛВЧ: 1,00
«Храброе сердце» (Х) «Разум и чувства» (РЧ) «Отверженные» (О)	ЛВЧ: –0,21, ЭВ: 0,01 Х: –0,23, О: 0,96 Х: –0,32, РЧ: 0,96	 О: 0,96 РЧ: 0,96

Как видно из таблицы, в окрестности у «Храброго сердца» в методе топ-*N* оказываются непохожие элементы, а метод порогового значения вообще не нашел похожих. Если вы используете метод топ-*N*, в рекомедации попадет *N* элементов, даже если они не являются подобными. Если вы используете пороговое значение, рекомендации могут оказаться пустыми. Теперь, когда мы определились с параметрами для выбора окрестности, ваши соседи будут отсортированы, и вы сможете начать вычисления прогнозов.

8.7. Методы прогнозирования оценок

Наиболее распространенные методы расчета прогнозов оценок делятся на две категории: регрессия и классификация; опишем их на двух примерах. На рис. 8.8 приведен обзор двух методов. На нем специально не указаны значения сходства, к которым мы еще вернемся, когда будем вычислять прогнозы.

Первый метод попробуем объяснить на примере того, как спрогнозировать цену продажи дома, используя цены аналогичных домов. Второй метод похож на подсчет голосов, как на выборах.

Стоимость дома (регрессия)

Чтобы понять суть метода регрессии, приведем пример с выбором цены продажи дома. Вы хотите выставить свой дом на продажу, но не знаете, сколько за него просить. Как вариант, вы можете найти похожие дома, которые тоже продаются (сопоставимые с точки зрения недвижимости), и поставить в среднем такую же цену. Этот метод похож на регрессию.

Предсказание оценок элементов для конкретных пользователей осуществляются таким же образом. Нам точно так же требуется найти похожих пользователей и усреднить их оценки элемента.

Дружелюбные избиратели (классификация)

Теперь представим, что у вас в городе есть 10 человек, которые баллотируются на дожность мэра. Вы не знаете, за кого голосовать, поэтому идете и спрашиваете у соседей, за кого планируют голосовать они. Затем вы считаете, сколько соседей планирует голосовать за каждого кандидата. Затем выбираете кандидата с наибольшим количеством голосов.

Этот метод легко переносится на наш случай. Вместо того чтобы просить соседей выбрать мэра, вы просите их оценить фильм. Гипотетически, если у активного пользователя в окрестности есть пять соседей и они хотят узнать, какую оценку поставить фильму, они посмотрят на оценки от других пользователей и подсчитают все их.

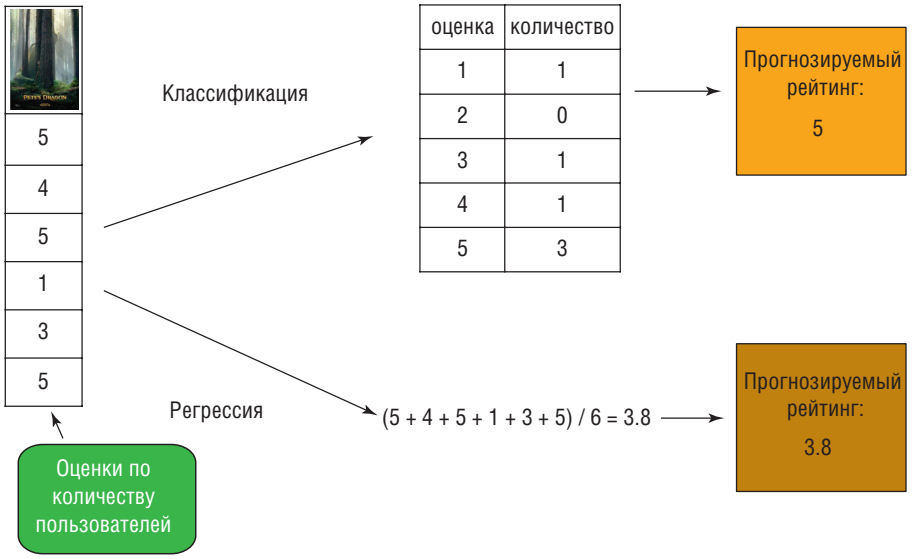


Рис. 8.8. Два способа расчета прогноза оценки фильма «Пит и его дракон» для активного пользователя, чьи соседи поставили оценки, указанные в списке справа. Классификация дает оценку 5, так как таких оценок больше, а регрессия дает 3,8, так как берет среднее из всех оценок

Теперь активный пользователь может посмотреть на число оценок и выбрать такую оценку, каких больше в его окрестности. Вместо того чтобы считать каждого соседа за один голос, вы можете использовать расчет сходства и учитывать только подобные голоса.

Это упрощенные примеры, в которых у оценок нет весовых коэффициентов. В следующем разделе мы более подробно поговорим о составлении прогнозов с использованием метода регрессии. Иллюстрация метода приведена на рис. 8.9. У вас есть активный пользователь, который оценил список элементов, и для каждого из этих элементов нужно найти подобные элементы в окрестности. Эти аналогичные элементы являются кандидатами на рекомендацию.

8.8. Прогнозирование с фильтрацией по элементам

Итак, мы подходим к самой сути. Вы создали список подобных элементов, изучая окрестности вокруг оцененных пользователями элементов, и теперь вы готовы составлять предсказания. Как это работает? Читайте дальше, и все станет ясно.

Теоретически можно взять все элементы в вашем каталоге и рассчитать прогнозы для них, но, поскольку вы основываете прогнозы на элементакх, которые активный пользователь уже оценил, нам нужны только элементы в этой окрестности.

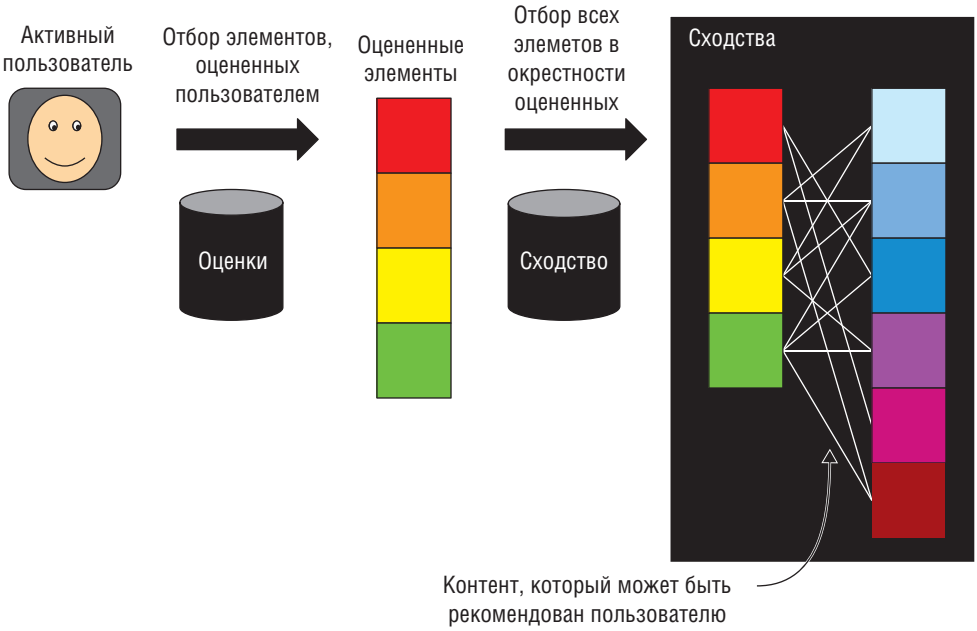


Рис. 8.9. Элементы, которые становятся кандидатами на рекомендаци, находятся в окрестности оцененных пользователем элементов.

Возвращаясь к примеру из раздела 8.7, мы более подробно рассмотрим, как предсказать оценку элемента.

8.8.1. Вычисление прогнозов

Когда вы найдете окрестность вокруг элемента, можно будет вычислить их оценку с помощью средневзвешенных оценок этих самых элементов из окрестности. В этом случае вам нужно добавить среднюю оценку пользователя, чтобы прогноз вычислялся в том же масштабе, что и оценки самого пользователя. С точки зрения математики мы получим следующую формулу:

$$Pred(u,\ i) = \overline{r}_u + \frac{\sum_{j \in S_i} \left(sim(i,\ j) \times r_{u,\ j} \right)}{\sum_{j \in S_i} sim(i,\ j)},$$

где:

- r_u – средняя оценка пользователя u;
- $r_{u,j}$ – оценка элемента j от пользователя u;
- S_i – набор элементов в окрестности оцененных пользователем элементов;
- $Pred(u,i)$ – прогноз оценки элемента i от пользователя u;
- $sim(i,j)$ – сходство между элементами i и j.

Допустим, вы хотите предсказать оценку Хеллы для «Стартрека»[1]. Сначала вы смотрите на вычисленную ранее модель сходства и берете фильмы, у которых окрестность получилась одинаковой независимо от того, какой метод вы использовали для ее определения. Получится список: СТ: ЛВЧ: 0,63, ЭВ: 0,35. Остается только подставить значения в функцию:

$$Pred(Хелла,\ Стартрек) = 2,6 + \frac{0,63 \times 0,4 + 0,35 \times 0,4}{0,63 + 0,35} = 3.$$

Если выполнили расчет примера в этой главе, можете поздравить себя, потому что вы рассчитали свой первый прогноз «элемент–элемент». Но подождите! Это же был прогноз оценки. Как превратить его в рекомендации? А очень просто. Этот расчет выполняется для всех элементов, представляющих интерес, затем элементы сортируются по их оценкам и выводится 10 лучших из них. Но это сработает только в том случае, если удастся найти 10 элементов для прогнозирования оценки. Если вы использовали метод порогового значения, предсказать оценку для «Храброго сердца» не удастся.

Следующая проблема заключается вот в чем: как рекомендовать фильмы, если ваш алгоритм дает им оценки ниже среднего? Не отчаивайтесь. Если вы запутались, мы скоро посмотрим на пример кода, который сможет понять даже машина.

8.9. Проблема холодного старта

Для совместной фильтрации требуются данные, которые трудно получить при появлении новых пользователей и новых элементов. В итоге у вас нет данных для выработки рекомендаций. Опять же, чтобы обойти эту проблему, можно

[1] Она уже есть таблице, но давайте все равно попробуем предсказать.

попросить новых пользователей оценить несколько элементов сразу при их появлении. Или, как вариант, можно дать новым пользователям посмотреть новый контент, многим это понравится. Мы подробно обсуждали это в главе 6. Или же вы могли бы использовать методы исследования, показанные на рис. 8.10.

На рисунке видно, что добавление новых элементов занимает 1/6 времени. Бросив кубик (или с помощью какой-то библиотеки генерации рандома), вы получите число, которое попадает в интервал *Использования* (1 – 5) или *Исследования* (6). *Использование* означает, что вы берете имеющееся знания для вывода рекомендаций. *Исследование* означает вывод нового элемента. Таким образом, элементы не подвергается риску, и пользователи, хочется верить, будут взаимодействовать с ними.

8.10. Пара слов о терминах машинного обучения

Мы собираемся реализовать алгоритм совместной фильтрации элементов. Сперва давайте определимся с терминами.

Плюс фильтрации элементов заключается в том, что большую часть тяжелой работы можно сделать заранее, до выдачи рекомендаций для активных пользователей. Если вы новичок в мире машинного обучения и Data Science, вы можете не знать, что эта самая работа называется *автономным обучением модели*. Давайте рассмотрим каждое из этих слов:

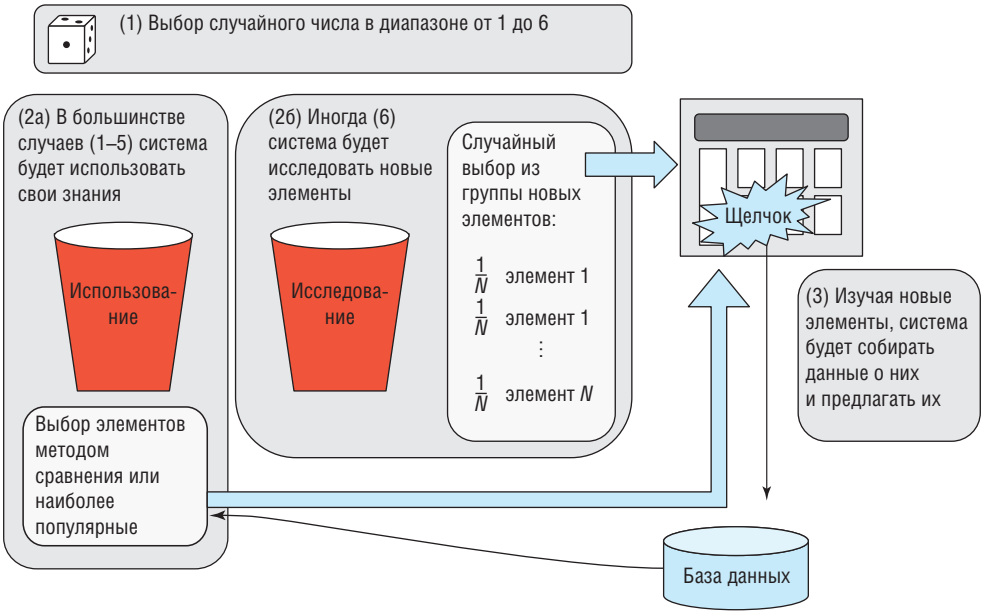


Рис. 8.10. Способ внедрять новые элементы в рекомендации с некоторым шансом

- *автономное* – выполняется офлайн, до использования в реальной работе, а не во время ожидания пользователя (в противовес *онлайн* – т. е. вживую);

- *модель* – средство для прогнозирования оценок (или других вещей в зависимости от алгоритма машинного обучения). Модель может быть создана путем взятия необработанных данных, их обработки через алгоритм машинного обучения, который агрегирует данные и создает модель. Создание модели обычно выполняется в автономном режиме;
- *обучение* – термин, пришедший из мира ИИ, где под машинным обучением понимается придание новых навыков. В нашем случае рекомендатель обучается (рассчитывая модель) находить схожие элементы.

На рис. 8.11 показано, что происходит в автономном и онлайн-режиме при совместной фильтрации элемент–элемент.

Чтобы наши примеры были близки к реальной жизни, в следующем разделе мы рассмотрим пример сайта MovieGEEKs. Если вы еще не сделали это, загрузите сайт с GitHub и следуйте инструкциям, приведенным на странице *README.md* по адресу **mng.bz/04k5**. Там же приведена инструкция, как загрузить и импортировать данные.

8.11. Совместная фильтрация на сайте MovieGEEKs

Мы будем использовать рекомендации после совместной фильтрации во втором ряду рекомендованных элементов (рис. 8.12), где, как вы помните, перечислены элементы, подобные тем, которые пользователь уже купил или оценил. Давайте начнем.

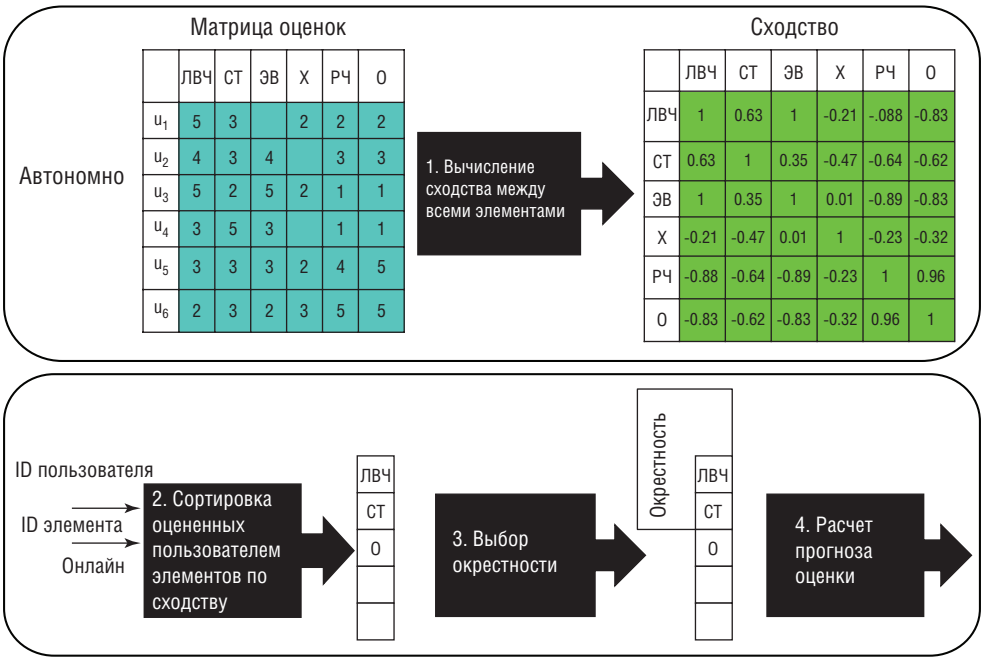


Рис. 8.11. Процедура совместной фильтрации разделена на офлайн- и онлайн-расчеты

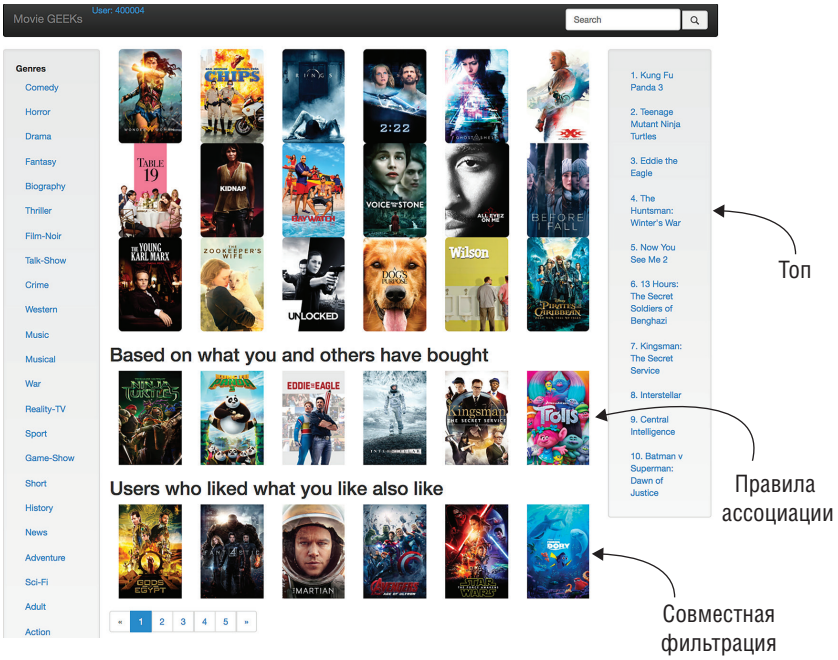


Рис. 8.12. Главная страница сайта MovieGEEKs с результатами совместной фильтрации

8.11.1. Фильтрация элементов

Надеюсь, после прочтения этой главы вам стало совершенно ясно, что нужно делать. Если нет, то вот список вещей, которые вы реализуете: (Они также показаны на рис. 8.11.)

- найти элементы, похожие на те, что понравились активному пользователю;
- рассчитать прогноз оценок этих элементов;
- использовать прогноз для расчета рекомендаций.

> **ПРИМЕЧАНИЕ.** Разница между совместной фильтрацией по пользователям и элементам заключается в методах определения сходства.

Если вы хотите просмотреть код, нужно заглянуть в два файла:

- *\builder\item_similarity_calculator.py* – код автономного обучения, где показаны все расчеты сходства;
- *\recs\neighborhood_based_recommender.py* – онлайн-часть .

Автономный расчет сходства

Для выполнения совместной фильтрации нам требуется матрица сходства, которая будет представлять собой таблицу элементов с рассчитанными значениями сходства. Для построения модели будет лучше создать сценарий Ру-

thon для запуска вне Джанго. Это может занять несколько часов, но, по крайней мере, не дней.

Вам нужно создать еще один билдер, как показано в листинге 8.1. Он получает данные из базы данных, вычисляет сходство и сохраняет значения сходства в базе данных.

ЛИСТИНГ 8.1. Использование сценария Python для создания и сохранения модели сходства

```
import os
import pandas as pd

import database
import item_cf_builder

all_ratings = load_all_ratings()

ItemSimilarityMatrixBuilder.build(all_ratings)
```

Получение данных оценок

Построение модели сходства и ее сохранение в базу данных

Извлечь оценки из базы данных просто, но я все равно рекомендую вам взглянуть на код. Если вы используете его, не забудьте добавить ограничение по времени для запроса, чтобы собирать только данные за определенный период времени. Я оставил его здесь, а также на Git (метод называется `load_all_ratings`). Давайте посмотрим на метод `build`, который будет обучать/строить/считать/выводить/варить модель в листинге 8.2. Как уже упоминалось, вы найдете метод в файле */builder/item_similarity_calculator.py*.

ЛИСТИНГ 8.2. Создание матрицы сходства

```
    def build(self, ratings, save=True):
        ratings['rating'] = ratings['rating'].astype(float)
        ratings['avg'] = ratings.groupby('user_id')['rating'] \
                                .transform(lambda x: normalize(x))

        ratings['user_id'] = ratings['user_id'].astype('category')
        ratings['movie_id'] = ratings['movie_id'].astype('category')

        coo = coo_matrix((ratings['avg'].astype(float),
                          (ratings['movie_id'].cat.codes.copy(),
                           ratings['user_id'].cat.codes.copy())))

        overlap_matrix = coo.astype(bool).astype(int) \
            .dot(coo.transpose().astype(bool).astype(int))
```

Оценки должны быть дробными, иначе будут проблемы

Нормализация оценок по средней оценке пользователя и ее приближение к данным. Используется метод `normalize`

Превращает `user_id` и `movie_id` в категории. Это нужно для использования разреженной матрицы

Превращает оценки в матрицу с названием `coo_matrix`[1]

Матрица перекрытия

[1] Более подробная информация приведена по ссылке **mng.bz/nLOL**.

```
    cor = cosine_similarity(coo, dense_output=False)

    cor = cor.multiply(cor > self.min_sim)

    cor = cor.multiply(overlap_matrix > self.min_overlap)

    movies = dict(enumerate(ratings['movie_id'].cat.categories))
    if save:
      self.save_similarities(cor, movies)

      return cor, movies

def normalize(x):
    x = x.astype(float)
    x_sum = x.sum()
    x_num = x.astype(bool).sum()
    x_mean = x_sum / x_num

    if x.std() == 0:
          return 0.0

      return (x - x_mean) / (x.max() - x.min())
```

Вычисление сходств между строк ми

Сохр нение в б зу д нных

Определение метод `normalize`

Сохр няет слов рь `movie_id` для поиск нужных элементов

Уд ление сходств с недост - точным перекрытием

Уд ление слишком низких зн чений сходств

Пару слов для предостережения[1]. Этот код был изменен и оптимизирован по сравнению с оригинальной версией, так что он работает довольно быстро. Вы можете запустить его и наслаждаться магией. Однако, даже если это быстрый код, его выполнение займет некоторое время.

Удаление сходства с недостаточным перекрытием

Мы говорили о том, насколько важно иметь достаточное количество дублирующихся оценок между двумя элементами. Но во многих реализациях методов сходства определить минимальное количество перекрывающихся элементов невозможно.

Покажу вам способ, который хорошо работает с разреженными матрицами. Вы берете матрицу оценок, превращаете ее в логическую (все, что больше нуля, превращается в `True`, а остальное – в `False`), а затем превращаем все в целые числа: 1 для `True` и 0 для `False`. Теперь вы можете умножить матрицу на нее же транспонированную, и получится матрица перекрывающихся элементов.

[1] Вам потребуется время, терпение и много свободной оперативной памяти. Когда я впервые придумал версию этого кода (будучи оптимистом, я думал, что это окончательный вариант), на выполнение ушло два дня. Эта версия экономит мне время, чтобы сходить погулять с собакой и пообедать.

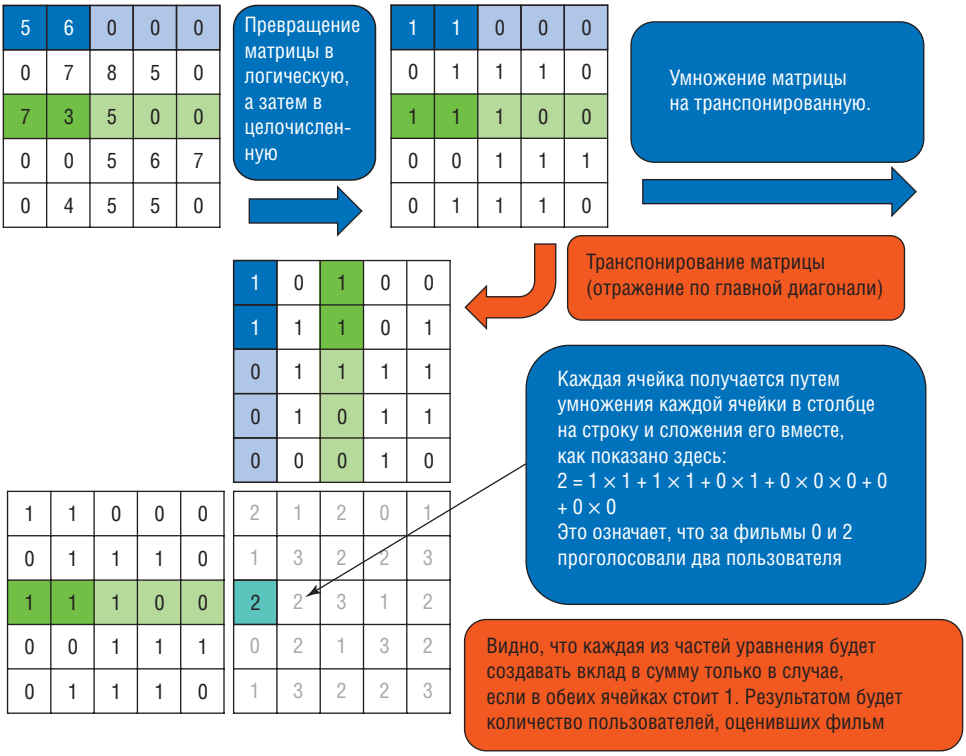


Рис. 8.13. Самый быстрый способ расчета перекрытия в вашей матрице оценок

Это показано на рис. 8.13 и реализовано в листинге 8.3 (фрагмент кода из листинга 8.2). Вы найдете этот код в файле */builder/item_similarity_calculator.py*.

ЛИСТИНГ 8.3. Создание матрицы перекрытия

```
overlap_matrix = coo.astype(bool).astype(int)
.dot(coo.transpose().astype(bool).astype(int))
```

Имея матрицу перекрытия, вы можете назначить каждой ячейке значение `True`, если значение в ней превышает предварительно определенное минимальное перекрытие. Если вам нужно перекрытие более 2 раз – назначьте этим ячейкам значение `True`, а остальным – `False`. Снова превратив матрицу в целочисленную, вы получите единицы там, где достигнуто нужное перекрытие. Это можно сделать с помощью кода, как показано в */builder/item_similarity_calculator.py* и листинге 8.4.

ЛИСТИНГ 8.4. Создание пороговой матрицы

```
overlap_matrix > self.min_overlap
```

Это выражение возвращает новую логическую матрицу. Когда вы рассчитали сходство, выполняется поэлементное умножение, а это значит, что вы умножаете каждую ячейку на другую ячейку с тем же индексом. Умножая матрицу сходства на логическую матрицу перекрытия, вы обнулите не интересующие вас значения сходства, а остальные не изменятся. В следующем листинге используется тот же прием для удаления слишком малых значений сходства (или см. */builder/item_similarity_calculator.py*).

ЛИСТИНГ 8.5. Удаление значений сходства

```
cor = cor.multiply(cor > self.min_sim)
cor = cor.multiply(overlap_matrix > self.min_overlap)
```

Уд ляет сходств , которые слишком м лы

Уд ляет сходств , которые не основ ны н дост точно перекрыв ющихся рейтинг х

Онлайн-прогнозы

Для использования модели, созданной с помощью показанных ранее методов, вам потребуется алгоритм предсказания, показанный на рис. 8.14. Обратите внимание, что вы обращаетесь к базе данных дважды – один раз, чтобы получить оценки пользователя, и второй – чтобы получить значения сходства. На самом деле это можно оптимизировать – подсоединиться к данным в базе данных до извлечения, чтобы получить и сходства, и оценки.

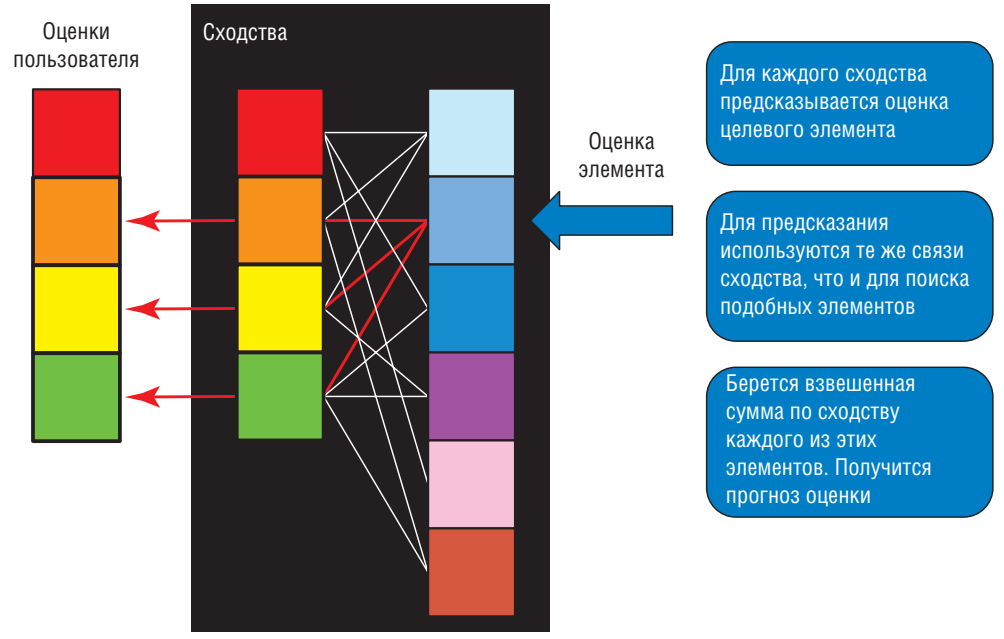


Рис. 8.14. Для прогнозирования оценок используется сходство между рассматриваемыми элементами и оцененными пользователем элементами. Все значения сходства умножаются на оценку, складываются и делятся на сумму сходств. Получается взвешенная сумма.

Если сценарий сходства выполнялся автономно, пришло время взглянуть на онлайн-часть.

ЛИСТИНГ 8.6. Код, выполняемый в момент захода пользователя на страницу

```
def recommend_items_by_ratings(self, user_id, active_user_items, num=6):

  movie_ids = {movie['movie_id']: movie['rating']
               for movie in active_user_items}
  user_mean = Decimal(sum(movie_ids.values())) / Decimal(len(movie_ids))

  candidate_items = Similarity.objects.filter(Q(source__in=movie_ids.keys())
                                              &~Q(target__in=movie_ids.keys()))

  candidate_items = candidate_items.order_by('-similarity')[:20]

  recs = dict()
  for candidate in candidate_items:
    target = candidate.target

    pre = 0
    sim_sum = 0

    rated_items = [i for i in candidate_items
                   if i.target == target][:self.neighborhood_size]

    if len(rated_items) > 1:
      for sim_item in rated_items:
        r = Decimal(movie_ids[sim_item.source]) - user_mean
        pre += sim_item.similarity * r
        sim_sum += sim_item.similarity
        if sim_sum > 0:
          recs[target] = {'prediction': user_mean + pre / sim_sum,
                          'sim_items': [r.source for r in rated_items]}

  sorted_items = sorted(recs.items(), key=lambda
                        item: -float(item[1]['prediction']))[:num]
  return sorted_items
```

Созд ние слов ря `movie_id` и оценок

Определение подобных оцененным элементов

Р счет среднего

Ост ется 20 зн чений из списк . С этим зн чением можно поигр ться, чтобы н йти б л нс между скоростью и к чеством

Перебор подобных элементов

Вычисление средней оценки для пользов теля

Умножение норм лизов нной оценки н сходство элемент

Приб вление сходств

Сортировк по прогнозу оценки. Здесь тоже можно доб вить в код что-то свое

Приложение прогноз оценки к результ ту вместе с другими оцененными элемент ми

Исключение деления н ноль

Жел тельно иметь более одного оцененного элемент , ин че прогноз будет т кой же, к к и его оценк

Просм трив ются сходств целевого элемент . Это д ет список элементов, оцененных ктивным пользов телем

Теперь увидите еще один список рекомендаций. Ну что? Рекомендации выглядят хорошо? Видите их на рис. 8.12? Откройте сайт от имени Хеллы и посмотрите на ее аналитику (**http://127.0.0.1:8000/analytics/user/400004/**). Вы увидите, что рекомендации довольно интересны (что показано на рис. 8.12)[1].

8.12. В чем разница между правилами ассоциации и совместной фильтрацией?

Давайте в последний раз взглянем на рекомендации, сравним их с рекомендациями, рассчитанными с помощью правил ассоциации и с помощью совместной фильтрации. Правила ассоциации, кажется, приближают к тому, что нравится Хелле. Но помните, что вы смотрите на два различных набора данных: правила ассоциации генерируются по покупкам ваших пользователей и автоматически генерируются скриптом, приведенным в главе 3, а совместная фильтрация работает на основе оценок в данных. В реальной системе это был бы один и тот же набор данных.

Ассоциативные правила и совместная фильтрация – это не одно и то же. В правилах используется информация о том, что пользователь купил сегодня, а не о том, что он покупает постоянно. А вот совместная фильтрация как раз учитывает долговременный период.

8.13. Эксперименты с совместной фильтрацией

Не всегда достаточно реализовать алгоритм, чтобы получить хорошие рекомендации. Часто есть вещи, которые приходится подкорректировать. Например, вы предварительно выбираете значение перекрытия пользователей, прежде чем вычислить сходство. Приведем список того, что вы можете подкорректировать:

- Какие оценки должны использоваться для активного пользователя:
  - только положительные?
  - только самые недавние из них?
  - как нормализовывать оценки?
- При вычислении сходства
  - сколько нужно пользователей, оценивших два фильма, для расчета сходства?
  - нужно ли ограничить сходство, например учитывать только положительное?
- При создании окрестностей
  - какой метод выбора окрестностей следует использовать?
  - насколько велика должна быть окрестность (значение порога или *N* для метода топ-*N*)?

[1] Чтобы добавить к пользователям персону, добавьте в запрос строковый параметр `user_id`. Чтобы увидеть страницы Хеллы (`user_id` которой `400004`), используйте следующий URL-адрес: **LOCALHOST: 8000/user_id = 400004**.

- При прогнозировании оценок
  - использовать классификацию или регрессию?
  - использовать ли средневзвешенное значение?
- При возврате рекомендации нужно ли возвращать только хорошие прогнозы оценок (выше некоторого порогового значения)?

И этот список можно продолжать и дальше.... На каждом этапе уменьшается количество элементов, для которых выполняются расчеты. Возьмем, к примеру, перекрытие. На рис. 8.15 показано, как перекрытие влияет на количество значений сходства в вашей модели для сайта MovieGEEKs. Если вам не требуется минимальное перекрытие, вы получите 40 млн значений. Если вам требуется два пользователя – останется 5 млн, и т. д.

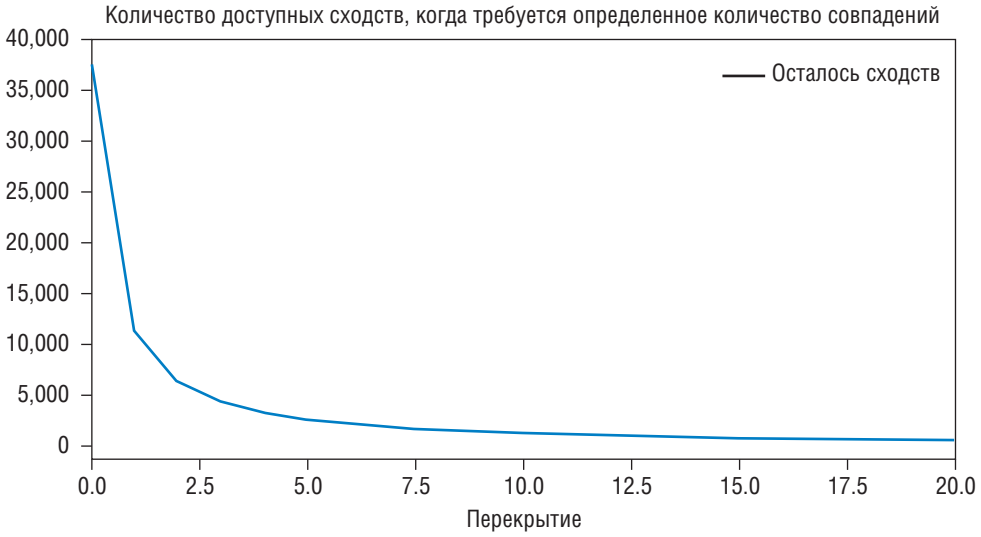


Рис. 8.15. График, показывающий значения сходства в тысячах при разных значениях перекрытия

На рис. 8.16 показан график, где фильмы поделены по количеству оценок, которые они получили. Около 11 000 фильмов были оценены одним человеком. Чтобы вычислить сходство между фильмами, вам нужно, чтобы их оценили два или больше пользователей, а это значит, что вы не можете использовать эти 11 000 фильмов для расчетов.

Посмотрев на строку выше 2000 на рис. 8.16, вы увидите, что есть около 4000 фильмов, которые были оценены двумя пользователями. Если установить предел перекрытия равным 2, то вы сможете убрать все, что слева. И если вам требуется по крайней мере две оценки, то около 12 600 фильмов будут отброшены. Это большое число, учитывая, что фильмов всего 25 000.

> **СОВЕТ.** Ряд сходств, сохраненных в базе данных, зависит от многих вещей, так что трудно определиться с жесткостью ограничений. Я советую не перебарщивать с этим, чтобы увидеть, как все работает, и затем при необходимости затянуть гайки. Вы найдете компромисс.

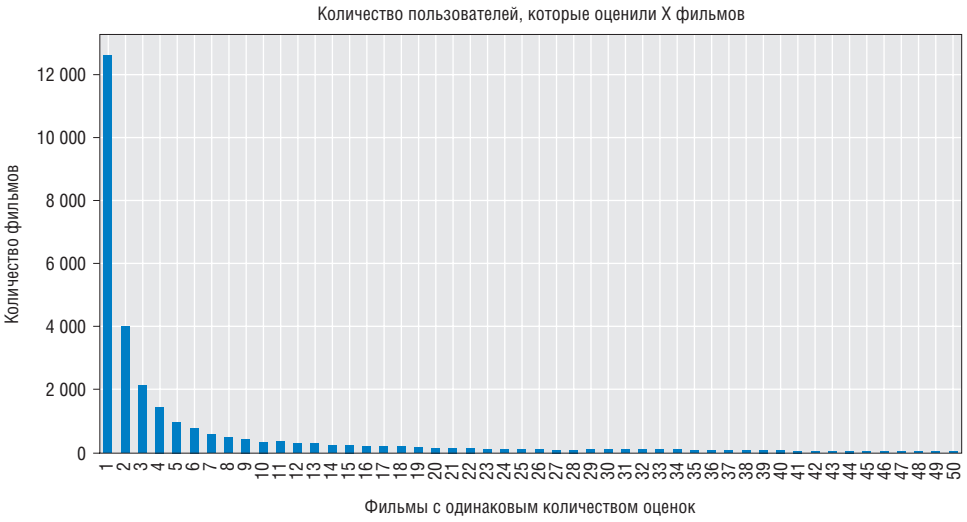


Рис. 8.16. График, который показывает, сколько фильмов получили *X* оценок на сайте MovieGEEKS

Как правило, чем больше пользователей оценили фильм, тем более он популярен, так как вам требуется больше перекрывающихся пользователей, а данные будут становиться менее персонифицированными. Рекомендации станут лучше, так как меньше фильмов будет в кандидатах на рекомендацию. По ходу главы и всей книги вы должны получить представление о том, что выбрать в каждом из предыдущих вариантов.

В главе 9 мы узнаем, как оценить рекомендательную систему. Прежде чем сломать всю голову при выборе правильных значений, прочитайте эту главу, а затем вернитесь к этому списку с учетом новых знаний. Чтобы знать, с чего начать, почитайте книгу Бадрула Сарвара «Рекомендательные алгоритмы совместной фильтрации по элементам»[1]. Но прежде чем перейти к главе 9, посмотрите на рис. 8.16, где показаны некоторые из функций оценок. Перед подведением итогов главы давайте перечислим плюсы и минусы алгоритмов совместной фильтрации в окрестности.

8.14. Преимущества и недостатки совместной фильтрации

Даже если это звучит нереально, у совместной фильтрации есть проблемы и недостатки, которые необходимо учитывать:

- разреженность: эта проблема является одной из самых серьезных. Большинство наборов данных недостаточно плотные, чтобы рекомендовать что-то, кроме наиболее популярных элементов. Поскольку большинство пользователей ставит всего несколько оценок, а в интернет-магазине

[1] Более подробная информация: **files.grouplens.org/papers/www10_sarwar.pdf**.

может быть много тысяч товаров , найти подобные элементы может оказаться непросто

- серые овцы: как вы помните из главы 6, серые овцы – это пользователи, вкусы которых столь необычны, что сложно найти для них рекомендации;
- количество оценок: как уже не раз упоминалось, для генерации хороших совместных рекомендаций требуется много оценок от пользователя, что позволит оценить его вкусы. Это проблема, поскольку большинство систем не может ждать, пока соберетеся 20 или больше оценок, и только потом выдавать рекомендации. Это та самая проблема холодного старта, о которой мы говорили в главе 6;
- сходство: совместная фильтрация безразлична к контенту и не пытается подгонять рекомендации под конкретные элементы. Рекомендации формируются по поведенческим тенденциям пользователей, которые всегда уделяют больше внимания более популярному контенту. То есть у популярного контента будет найдено сходство с большим количеством элементов, поэтому популярный контент будет рекомендоваться чаще.

Тот факт, что совместная фильтрация безразлична к контенту, – это в то же время большой плюс. Вам не придется тратить время на добавление в контент метаданных или на сбор информации о пользователях. Вам нужны только оценки и взаимодействие между элементами. В следующей главе мы узнаем, как можно оценивать рекомендательные системы.

Резюме

- В процедуре фильтрации в окрестностях можно использовать либо фильтрацию по пользователям (поиск подобных пользователей), либо по элементам (поиск подобных элементов).
- Стоит использовать фильтрацию по пользователям, если у вас элементов больше, чем пользователей. В противном случае нужно использовать фильтрацию по элементам
- Матрица сходства позволяет быстро найти подобные элементы.
- Использование таблицы сходства позволяет системе определить окрестности, применяя методы кластеризации, топ-*N* или пороговое значение.
- Найденные окрестности позволяют рассчитать прогнозы, если у вас есть небольшой набор подобных пользователей.
- На сайте Amazon использовалась рекомендательная система на основе совместной фильтрации по элементам.

Глава 9

Оценка и тестирование рекомендательной системы

Netflix Prize свели задачу рекомендации к задаче точного прогнозирования оценок. Теперь ясно, что это только один из многих компонентов эффективной рекомендательной системы. Нужно также учитывать такие факторы, как разнообразие, контекст, правдивость, свежесть и новизна.

Завьер Аматриан и др.[1]

Изучив эту главу, вы получите опыт в следующих областях:

- оценка эффективности алгоритма рекомендатора;
- разделение наборов данных на обучающие данные и тестовые данные;
- проектирование автономных экспериментов для оценки рекомендательных систем;
- общая информация об онлайн-тестированиях.

Зачем вы создаете рекомендатор? Что вы хотите получить? Больше денег? Больше посетителей? Попробовать новую технологию? Независимо от вашего желания его нельзя взять и превратить в вычисление, даже если вам нужна модернизация[2]. Мы часто слышим об алгоритмах, которые стали лучше по сравнению с текущими ультрасовременными алгоритмами, но в чем заключается это улучшение?

Эта глава посвящена оценке рекомендательных систем или, вернее, попытке оценить их. Исследователи рекомендательных систем сошлись во мнении о том, что почти невозможно оценить рекомендательную систему или алго-

---

[1] Amatriain, Xavier et al., «Past, Present, and Future of Recommender Systems: An Industry Perspective» (Recsys, 2016).

[2] Книга «Start with Why: How Great Leaders Inspire Everyone to Take Action» (Portfolio; Reprint edition, 2011) Саймона Синека не имеет никакого отношения к рекомендательным системам, но интересна с точки зрения понимания того, зачем вам нужен ваш бизнес.

ритм, не имея живой системы, на которой его можно испытать. Тем не менее важно знать, движется ли ваша рекомендательная система хотя бы в правильном направлении.

9.1. Бизнесу нужен подъем, перекрестные продажи, рост продаж и конверсии

В этой главе мы изучим, как лучше оценить систему, используя имеющиеся данные. Мы также поговорим о том, как протестировать живой сайт. Вы, конечно, можете эмулировать посетителей и посещения, чтобы дать системе данные для запуска, но смысла в моделировании посещений для оценки рекомендатора нет.

На рис. 9.1 показан цикл оценки рекомендательной системы. Мы подробно остановимся на каждом шаге в течение этой главы. Некоторые задачи обычно не связаны с оценкой рекомендательных систем, но они важны, если вашей системе нужны развитие и поддержка.

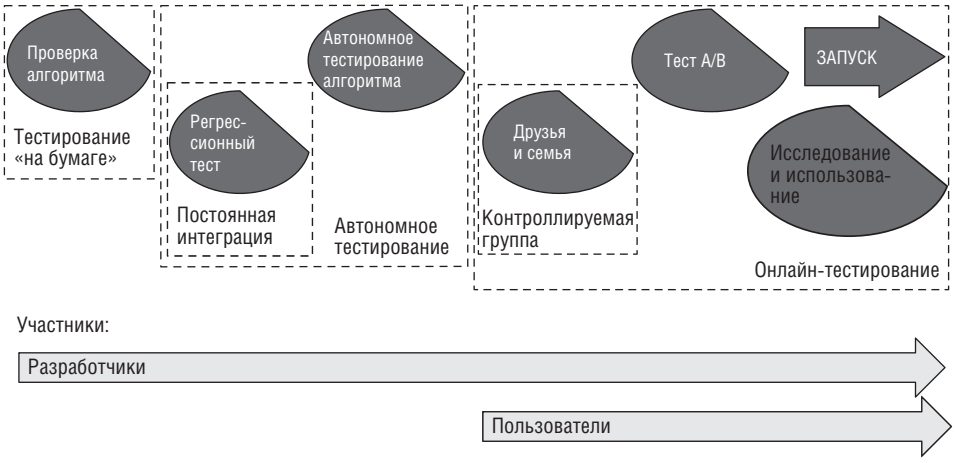


Рис. 9.1. Цикл оценки рекомендательных систем и участники процесса оценки

Часто забывают, что все системы, работающие с данными, – словно живые существа (не в смысле ИИ), и им требуется поддержка и мониторинг. Производительность – это продукт использования данных, которые постоянно обновляются. Поведение системы изменяется при изменении данных, а производительность и прогностическая мощность системы может тоже изменяться или ухудшаться.

Прежде чем слишком углубиться в детали, давайте рассмотрим задачи, показанные на рис. 9.1. Я рекомендую вам начать с простешего алгоритма, а затем наращивать сложность по мере необходимости. Когда вы выбрали алгоритм, сперва требуется проверить его.

Возьмите небольшой набор данных – например, 5 пользователей и 10 элементов, и посмотрите, будет ли работать ваш алгоритм. Поиграйте с ним, и,

если результат вам понравится, начинайте реализацию алгоритма. Хорошо бы проводить постоянные регрессивные тесты таким образом, чтобы видеть, что обновление данных вредит производительности системы. Когда пишется код алгоритма и тестируется его функциональность (непрерывно), можно выполнить автономную оценку алгоритма с использованием набора данных. Если оценка выполняется успешно, можно показывать результаты людям.

Часто сперва используется контрольная группа – как правило, близкие. И здесь придется объясняться перед боссом (который не читал эту книгу и ничего знать не знает о рекомендательных системах, но своим глазом эксперта может определить, что рекомендации какие-то неправильные и не подходят ему). Друзья и семья помогут только с тем, чтобы убедиться в правильности работы рекомендатора, но их будет мало, чтобы проверить ключевые показатели эффективности (KPI), которые в конечном счете вам и нужны. Если все прошло успешно, пришло время запускать алгоритм для всех.

Многие останавливаются на этом моменте, но сейчас популярна тенденция использования и исследования, т. е. вы позволяете системе решить, какой алгоритм следует применять, основываясь на том, как пользователи реагируют на результаты ее работы. Мы кратко поговорим об этом далее в этой главе. Да и вообще, мы много о чем поговорим, так что давайте углубимся. Прежде чем перейти к тому, как оценить ваш рекомендатор, давайте немного поговорим о том, зачем это вообще нужно.

9.2. Зачем оценивать?

Ваша рекомендательная система работает правильно? После того как с потом и кровью ваша рекомендательная система заработала, вы не хотите потерпеть неудачу. Часто тестирование рекомендательной системы – это реализация вашего желания найти доказательства того, что она работает. Вы также хотите *не* найти признаки неидеальной работы или выдачи неправильных результатов более чем половине пользователей. Было бы, конечно, хорошо, если бы вы узнали о таком раньше, чем все ваши пользователи!

Часто система чересчур подгоняется под предпочтения тех, кто у руля. Босс звонит и говорит: «Моей дочурке не нравится ни одна из рекомендаций вашей треклятой системы. Если не исправишь – собирай манатки.» И разумеется, он говорит это где-то в пятницу после обеда, а у вас, конечно же, были другие планы на выходные. Может, я слегка утрирую, но такую ситуацию надо принимать во внимание, так как разным людям нужны разные рекомендации.

Имея это в виду, вы должны четко представлять, какие ваши цели. Давайте вернемся к Netflix. Одной из целей платформы является сохранение числа абонентов, но оно измеряется лишь раз в месяц. Netflix вывел (возможно, изучая данные), что удержание пользователя коррелирует с его просмотрами. Но если пользователь смотрит больше контента, это может указывать и на другие вещи, и не обязательно долгосрочные. Может быть, пользователя бросила девушка и у него освободилась куча времени. Да мали ли что может быть! Кроме того, цель может оказаться неизмеримой, а измеримые вещи необязательно однозначно дадут вам все ответы.

Поскольку нам нужна только рекомендательная система сама по себе, наша оценка должна показывать, какой рекомендатор дает лучший результат на основании данных MovieTweetings[1], которые мы используем в нашем примере. Чтобы оценить работу на большом количестве людей (в случае MovieGEEKs человек получает рекомендацию), потребуется погрузиться в статистику. Лекций по статистике в этой книге не будет, но я расскажу вам, что вы должны иметь в виду при тестировании рекомендатора, который вы реализовали. Сперва представим нашу задачу в виде гипотезы.

Гипотеза

Гипотеза описывает цель тестирования. Для нас она может звучать так: «Рекомендатор Б дает рекомендации, на которые щелкали чаще, чем на рекомендации Рекомендатора А». Событие «щелчка» также называется частотой кликов, или CTR.

Достаточно ли гипотеза ясна, чтобы запустить тест? Поймет ли ее правильно почтальон, которому вы покажете эту гипотезу?

9.3. Как интерпретировать поведение пользователей

На рис. 9.2 показаны четыре сценария для посетителя, заходящих на сайт:

- *визит 1 – простой*. Ваша система записывает, что пользователь зашел на целевую страницу, а затем ничего не произошло. Есть ли смысл рассуждать о том, видел ли посетитель рекомендацию?
- *визит 2 показывает, что пользователь просмотрел несколько страниц*. Здесь вы должны предположить, что рекомендации появились, но посетитель не проявил к ним никакого интереса, так как он щелкал на что-то другое;
- *визит 3 – идеальный*. Посетитель нажимает на рекомендацию;
- *визит 4 ставит вопрос*. Как оценить этот визит? Считать его как один или два щелчка? Возможно, есть смысл говорить о том, что рекомендательный алгоритм отработал успешно, если пользователь щелкнет по рекомендациям один или несколько раз. Но опять же, это зависит от вашей задачи.

9.4. Что измерять

Выделим два типа целей для рекомендательных систем:

- сделать посетителей довольными и заодно подзаработать деньжат;
- заработать как можно больше денег, и если для этого нужны счастливые пользователи – то сделаем и это, но только в достаточных пределах, чтобы заработать деньги.

[1] См. более подробную информацию: **github.com/sidooms/MovieTweetings.**

Вам, разработчику, нужно, чтобы был доволен босс, а он будет доволен, если клиенты довольны (и будут тратить у вас свои деньги). Цели клиентов примерно одинаковы: они хотят быть счастливы и будут готовы тратить деньги в этом случае, но, в отличие от босса, они хотят, чтобы затраты были поменьше.

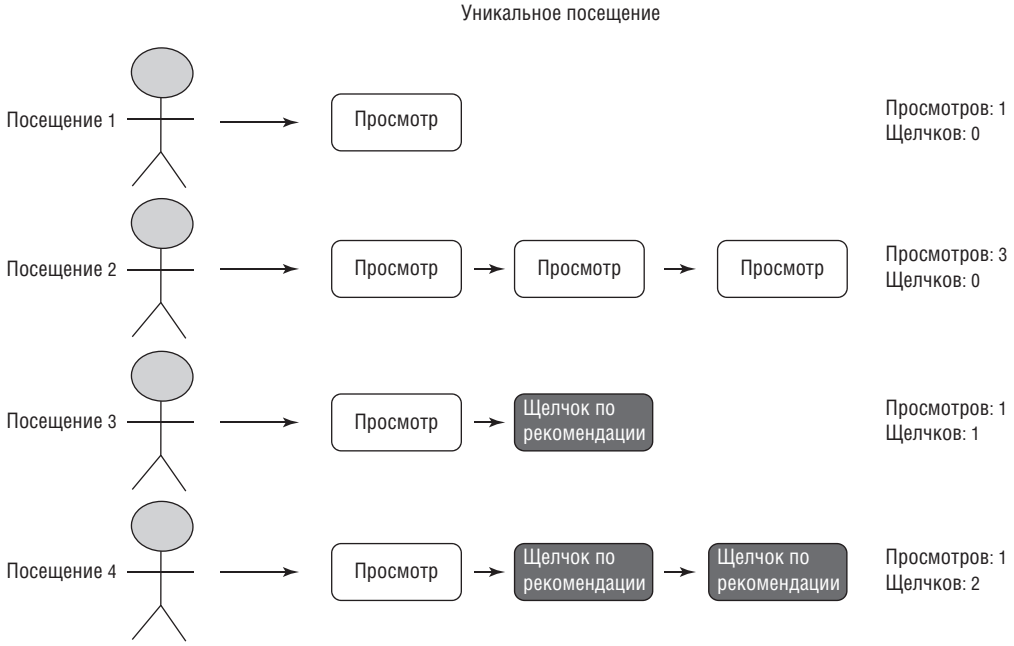


Рис 9.2. Различные сценарии поведения пользователя. Посещение 3 является идеальным, поскольку пользователь просматривает и щелкает рекомендацию

Давайте на минуту забудем о деньгах и подумаем о счастливых клиентах. Как вы можете сделать клиентов счастливыми? У каждого клиента и в каждой задаче ответ будет разный, но мне нравятся следующие вещи:

1. *Сайт понимает мои вкусы*. Я бы не хотел получать слишком уж неправильные рекомендации.
2. *Сайт дает мне разнообразные рекомендации.* Я хочу, чтобы мои рекомендации были разнообразны.
3. *Сайт удивляет меня*. Я хочу находить что-то новенькое, что мне неожиданно бы понравилось.

Для сайта или сопровождающего контента добавим еще одно:

4. Сайт охватывает весь свой каталог.

Мы остановимся на каждом пункте подробнее.

9.4.1. Понимание вкусов пользователя: сведение к минимуму ошибки предсказания

Один из способов измерить то, насколько хорошо рекомендатор понимает мои вкусы, – это дать ему предсказать мою оценку для элемента, который

я уже знаю и оценил. Таким образом, мы можем проверить, как часто рекомендатель предсказывает оценку близко к правильной.

Можно также воспользоваться *метрикой поддержки решений*, в которой прогнозы разделяются на группы по признаку их соответствия реакции пользователя. Если система предсказывает оценку фильма выше 7 по шкале от 1 до 10, то он, вероятно, будет интересным. Если она ниже 7, но выше 3, то я бы посмотрел этот фильм, но только если жена захочет, чтоб я составил ей компанию. В случае оценки ниже – даже не просите. Ни в коем случае. Если рекомендатор предсказывает оценку близко к той, которую я поставил бы я сам, значит, он работает хорошо. Если моя оценка была бы другой, это заставило бы меня зря потерять время или хороший контент. А если он будет часто ошибаться, я вообще перестану пользоваться рекомендациями.

9.4.2. Разнообразие

Давайте вспомним кое-что из Библии, а точнее из книги от Матфея:

> *Ибо всякому имеющему дастся и приумножится,*
> *а у неимеющего отнимется и то, что имеет;*
> 25:29 Матфея, версия короля Джеймса

Вы, вероятно, спросите: «А причем тут рекомендательные системы?» Библейский стих заложил основу того, что называется *эффектом Матфея*. Иными словами, «богатые становятся богаче, а бедные становятся еще беднее». Если рассматривать популярный элемент как богатого, а непопулярный – как нищего, то проблема будет в том, что в вашем каталоге у вас будут элементы, которые рекомендуются часто, так как они популярны.

Это хорошо, что популярные элементы рекомендуется часто, но в то же время они появляются в рекомендациях слишком часто, становясь еще популярнее, а другие потенциально хорошие фильмы в рекомендации уже не влезают и остаются в тени. Когда все смотрят только популярные фильмы, возникает явление под названием *информационный пузырь*[1]. Это, может быть, и хорошо, так как вы всегда будете получать в рекомендациях то, что вам нравится. Но с другой стороны, вы так никогда и не узнаете, что существуют и менее популярные фильмы, которые вам тоже понравятся.

В дополнение к проблеме пузыря есть также тот факт, что первоначальная идея рекомендательной системы состоит в том, чтобы помочь пользователям ориентироваться в больших каталогах, а не ограничиваться одной страницей магазина. Если рекомендатор показывает только популярные элементы, это преимущество теряется (есть много других преимуществ, но это важное).

Трудно измерить, насколько разнообразно работает система, а системе трудно оставаться разнообразной, если она персонализирована.

Исследователи пытались вычислить меру разнообразия путем вычисления среднего несходства между всеми парами рекомендуемых элементов. При-

[1] Более подробная информация: **en.wikipedia.org/wiki/Filter_bubble**.

мер – книга «Improving Recommendation Diversity» Кита Брэдли и Барри Смита. Вы можете найти статью по адресу **www.academia.edu/2655896/Improving_recommendation_diversity**.

9.4.3. Охват

Разнообразие тесно связано с охватом, поскольку чем лучше разнообразие, тем шире охват. Одна из главных задач внедрения рекомендательной системы – позволить пользователям перемещаться по всему каталогу контента. Широкий охват означает, что алгоритм будет рекомендовать все элементы в каталоге. Способность рекомендовать что-то для всех зарегистрированных пользователей называется *охватом пользователей*.

Самый прямолинейный способ вычисления охвата пользователей – это перебрать всех пользователей, вызвать алгоритм рекомендатора, а затем посмотреть, возвращает ли он результат. Пример кода для этой задачи показан в листинге 9.1. Можно просмотреть сценарий для этого метода в папке */evaluator/coverage.py*.

ЛИСТИНГ 9.1. Расчет охвата

```
def calculate_coverage(self):

  for user in self.all_users:                    <-- Перебор всех пользов телей
    user_id = str(user['user_id'])
    recset =
     self.recommender.recommend_items(user_id)   <-- Генер ция рекоменд ций
    if recset:                                   <-- Проверк того, получен ли рекоменд ция
      self.users_with_recs.append(user)          <-- Доб вление пользов теля в лист получивших рекоменд ции
    for rec in recset:                           <-- Вывод всех рекомендов нных элементов
      self.items_in_rec[rec[0]] += 1             <-- Доб вление элементов в рекомен-д цию

  no_movies = Movie.objects.all().count()
  no_movies_in_rec = len(self.items_in_rec.items())
  no_users = self.all_users.count()
  no_users_in_rec = len(self.users_with_recs)
  user_coverage = float(no_users_in_rec/ no_users)     <-- Р счет охв т
  movie_coverage = float(no_movies_in_rec/ no_movies)  <-- Р счет охв т элементов
  return user_coverage, movie_coverage
```

Этот метод рассчитывает охват и по элементам, и по пользователям. Охват пользователей вычисляется как

$$охват_{пользователь} = \frac{\sum_{u \in U} P_u}{|U|},$$

где:

$$P_u = \begin{cases} 1 & : if \left|recset\right| > 0, \\ 0 & : else \end{cases}$$

$|U|$ = количество всех пользователей,

$|recset|$ = количество рекомендаций, которые рекомендатель возвращает пользователю u.

Используя результат выполнения метода, вы можете также рассчитать охват каталога, который определяется по формуле:

$$охват_{каталог} = \frac{\left|все\ элементы\ в\ recset\right|}{\left|I\right|},$$

где $|I|$ = *количество элементов в каталоге.*

Для расчета охвата вам также потребуется знать общее количество элементов в каталоге. С помощью метода `calculate_coverage` из листинга 9.1 я обнаружил, что совместная фильтрация по элементам, реализованная в главе 8, дает охват 13 %. Из 26 380 фильмов в наборе данных MovieGEEKs только 3473 фильма могли попать в рекомендации[1].

Прогоняя этот код, я также выяснял, какие фильмы попадали в рекомендации. Я получил гистограмму, показанную на рис. 9.3. На ней видно, сколько фильмов было рекомендовано только один раз среди всех рекомендаций, подготовленных для всех пользователей. Наиболее популярные фильмы находятся в длинном хвосте графика (что немного нелогично по сравнению с проблемой длинного хвоста, о которой мы говорили ранее). График обрезан справа, потому что есть фильмы, которые появились в более чем 100 наборах рекомендаций.

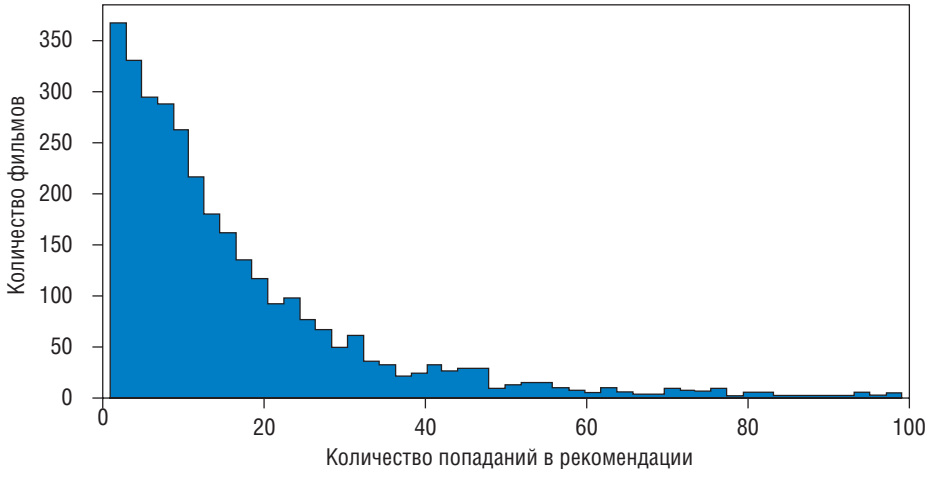


Рис. 9.3. Сколько фильмов показывалось *X* раз в рекомендации. Более 350 фильмов были показаны только в одной рекомендации. При этом наиболее популярные фильмы находятся в длинном хвосте

[1] Это зависит от того, какие параметры вы используете на этапе обучения.

Обратите внимание, что 3473 фильма из 26 380 – это маловато. Может быть, стоит добавить рекомендатор на основе контента, о котором мы поговорим в следующей главе. Код, приведенный в листинге 9.1, можно запустить с помощью строки, показанной в следующем листинге.

ЛИСТИНГ 9.2. Запуск алгоритма охвата в алгоритме совместной фильтрации

```
>python -m evaluator.coverage -cf
```

Если вы выполните команду `--help` (как показано в следующем листинге), вы увидите, что у вас есть возможность запуска оценки охвата в каждом из алгоритмов, которые мы рассмотрим в книге далее.

ЛИСТИНГ 9.3. Справка по алгоритму совместной фильтрации

```
Evaluate coverage of the recommender algorithms.

optional arguments:
  -h, --help  show this help message and exit
  -fwls run   evaluation on fwls rec
  -funk run   evaluation on funk rec
  -cf run     evaluation on cf rec
  -cb run     evaluation on cb rec
  -ltr run    evaluation on rank rec
```

Но не запускайте остальные функции, пока ваши модели не обучены. В противном случае вычисление охвата займет много времени.

9.4.4. Приятные неожиданности

Вам хочется неожиданно находить в рекомендациях вещи, которые вы, сами того не зная, любите. То есть сайт должен давать пользователю именно такое ощущение, что «нельзя зайти дважды на один и тот же сайт». Это штука субъективная, и ее трудно рассчитать, поэтому я прошу вас помнить, что важно не слишком сильно ограничивать свободу рекомендатора. Ограничения снижают шанс получения сюрпризов. Вы можете прочитать о попытках применить метрики неожиданности в статье «В сторону от точности: оценка охвата и случайности рекомендаторов»[1].

Теперь вы знаете несколько параметров, которые нужно измерять и оценивать. Мы стали на один шаг ближе к оценке вашей рекомендательной системы, но сначала, как бывший разработчик программного обеспечения, я обязан вам дать кое-что еще.

[1] Подробнее: **mng.bz/nkGC**.

9.5. Перед реализацией рекомендатора...

Еще одна пословица: «Лампа горит только с налитым маслом». Поэтому нам нужно рассмотреть следующие вопросы:

- проверка алгоритма;
- проверка данных;
- выполнение регрессиионного теста.

Давайте рассмотрим их более подробно.

Проверка алгоритма

Вроде бы глупо говорить об этом, но вы удивитесь, как часто кто-то читает крутейшую научную статью, затрачивает месяцы на реализацию алгоритма – а потом внезапно кто-то спрашивает: «А зачем козе баян?» Слишком поздно они понимают, что этот алгоритм не будет работать с имеющимися данными и давать ожидаемые результаты. Упростите себе жизнь и напишите простой сценарий, в котором вы можете строго просчитать алгоритм на небольшом примере. Тщательно продумайте, какие данные вы можете предоставить в качестве входных, и обязательно согласуйте с заинтересованными сторонами выходные результаты.

Этот простой сценарий также поможет вам убедиться, что система работает. Например, алгоритм, который мы рассмотрим в этой главе, на полном наборе данных выполняется несколько часов. Если использовать такие данные для отладки алгоритма, писать его придется до пенсии, так как внеся одно исправление, придется запускать весь процесс заново. Если вам не повезло, на следующем шаге придется снова что-то исправлять. Упростите себе жизнь и следуйте инструкциям, приведенным на рис. 9.4.

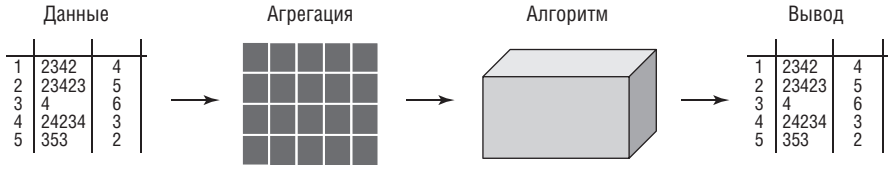


Рис. 9.4. Просчитайте алгоритм вручную, чтобы проверить, что у вас есть данные и правильность его вывода

Данные

Проверьте, есть ли у вас нужные данные или можно ли их создать. Подойдут ли для использования данные, сгенерированные в далеком прошлом? Это замечательно, если вы сохраняете оценки и поведение всех пользователей, но, если система удаляет устаревшие данные, могут возникнуть проблемы.

Также стоит взглянуть на разнообразие данных. Есть ли у вас данные пользователей, которые взаимодействуют со всем вашим каталогом, или есть данные только для его части? Потребуются дополнительные действия, чтобы задействовать эти элементы.

Архитектура и агрегирование

Наличия данных недостаточно. Можно ли получить их? Если да, то можно ли агрегировать нужные данные? В SQL базах легко настроить связи между таблицами, но, например, для MongoDB это не подходит. Это кажется очевидным, но нужно выполнить шаг, прежде чем полагаться на данные.

Алгоритм

Насколько сложен алгоритм? Легко ли его реализовать? Нужна ли ему сложная математика, которую трудно реализовать или которая требует слишком много вычислительной мощности? А если создать небольшой пример, используя небольшой набор данных, можно ли сделать наивную реализацию, с которой справится даже слабый компьютер? Когда эти вопросы будут решены, вы можете приступить к реализации вашего алгоритма.

9.5.2. Регрессионное тестирование

Как инженер-программист, вы должны знать о регрессионном тестировании. А это означает, что у вас должен быть набор тестов, которые можно запускать либо по ночам, либо по крайней мере каждый раз, когда кто-то делает изменения в коде. Часто люди утверждают, что их алгоритмы слишком сложны для выполнения автоматических тестов. Если вы столкнетесь с одним из этих людей, пожалуйста, приложите его чем-нибудь, особенно если он утверждает, что его бизнес не может себе это позволить.

Возьмем, к примеру, алгоритм совместной фильтрации. Он построен как процедура с несколькими этапами. Каждый из этапов можно проверить отдельно. Например, метод сходства можно легко протестировать, посмотрев, правильно ли он реагирует на простые векторы. При вызове метода сходства с совпадающими векторами схожесть должна равняться 1, а для перпендикулярных методов – 0 или –1 в зависимости от того, какая используется функция сходства.

Посмотрите на тесты, которые я сделал в тестовой папке проекта[1]. Одна из функций сходства выглядит так же, как показано в следующем листинге. Опять же, вы можете просмотреть код в сценарии */item_similarity_calculator_test.py*.

ЛИСТИНГ 9.4. Тест функции сходства

```
def test_simple_similarity(self):
    builder = ItemSimilarityMatrixBuilder(0)

    no_items = len(set(self.ratings['movie_id']))
    cor = builder.build(ratings=self.ratings, save=False)
    self.assertIsNotNone(cor)
    self.assertEqual(cor.shape[0], no_items,
      "Expected correlations matrix to have a row for each item")
```

[1] Подробнее: **mng.bz/rS4s**.

```
    self.assertEqual(cor.shape[1], no_items,
      "Expected correlations matrix to have a column for each item")
    self.assertEqual(cor[WONDER_WOMAN][AVENGERS], - 1,
      "Expected Wolverine and Star Wars to have similarity 0.5")
    self.assertEqual(cor[AVENGERS][AVENGERS], 1,
      "Expected items to be similar to themselves similarity 1")
```

Я создал небольшой набор данных для тестирования, который позволяет мне запускать небольшие тесты и убедиться, что каждый этап совместной фильтрации работает правильно. И пока мы тут, пришло время посмотреть на оценку в автономном режиме, с которой обычно начинают разговор об оценке машинного обучения и рекомендательных алгоритмов.

9.6. Виды оценки

Есть несколько способов протестировать рекомендательный алгоритм. Не все они могут дать вам точное представление о том, как будет работать алгоритм на реальном сайте. Печальная истина состоит в том, что единственный реальный способ узнать это – запустить алгоритм на своем сайте. Но, прежде чем сделать это, вы можете облегчить переход, выполнив кое-какие оценки.

Мы предполагаем, что у вас уже есть набор данных оценок. Для правильной оценки нужно вычислить так называемый *полный эталонный набор*, который представляет собой набор данных, содержащий информацию обо всех комбинациях пользователей и контента. При наличии такого набора вам, возможно, и не понадобится рекомендательная система, потому что пользователь уже и так составил свое мнение обо всех элементах. Если у вас нет такого набора, вам нужно вместо этого предположить, что сочетания пользователей и элементов, присутствующие в этих данных, правдивы для всех пользователей.

У вас есть три типа сценариев проверки: офлайн-эксперимент, контролируемый эксперимент с пользователями и онлайн-эксперимент. Исследователи рекомендательных систем считают, что офлайн-эксперименты не работают, а контролируемые и онлайн-эксперименты слишком затратны, но всем не угодишь. Шон МакНи и его коллеги из GroupLens[1] отмечают, что упор на повышение точности рекомендательных систем был ошибочным и иногда даже вредным. У каждого эксперимента (офлайн, с пользователями или онлайн) своя цель. Вам нужно только выбрать нужный вам вариант.

9.7. Офлайн-оценка

Идея автономной или офлайн-оценки заключается в использовании готовых данных, которые вы считаете правдивыми. Затем требуется разделить данные

[1] Шон М. МакНи и др. «Точность – не всегда хорошо: как метрики точности вредят рекомендаторам», **files.grouplens.org/papers/mcnee-chi06-acc.pdf**.

на две части и скормить одну часть рекомендатору. Другую часть вы можете использовать, чтобы проверить, как рекомендатор предсказывает оценки элементов. Прогнозы, близкие к реальным оценкам, или те, которые позволяют составлять рекомендации правильно, показаны на рис. 9.5. Или вы можете сказать, что пользователи смотрят элеметы, которые они оценили, даже если оценка была не очень хорошей, – и эти оценки тоже имеют значение.

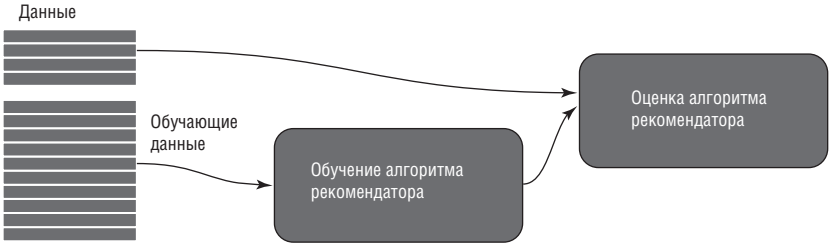


Рис. 9.5. Для офлайн-оценки данные перед тестированием алгоритма разделяются на обучающие и тестовые

Это считается не очень хорошим способом оценки рекомендаторов, но придумать лучший способ трудно, если вы не Netflix, где A/B тестирование всех новых функций выполняется вживую (об этом мы поговорим немного позже). Автономная оценка используется для измерения эффективности рекомендатора. Но бывает так, что рекомендатор может успешно пройти автономную оценку, а потом с треском провалиться в реальной работе.

Тут много сослагательного наклонения и предположений, но мы поработаем с рекомендатором, который рассмотрели в главе 8. В следующих главах мы поговорим о том, как оценить алгоритм. Если вы хотите исследовать рекомендательные системы, то лучше бы вам иметь хорошее понимание того, как реализовывать, тестировать и представлять новые алгоритмы или модернизации существующих. Я рекомендую вам посмотреть статью Майкла Экстранда и соавторов об экосистеме рекомендаторов[1].

9.7.1. Что делать, если алгоритм не дает рекомендаций

Часто вы будете сталкиваться с ситуациями, когда рекомендатор не выдает рекомендаций вообще или дает их мало. Это может быть проблемой во время оценки системы, поэтому можно заполнить пробелы данных с помощью простого алгоритма, выбирая наиболее популярные элементы, среднюю оценку для данного элемента или среднюю оценку пользователей. Несколько более сложным решением является использование базовых рекомендателей, которые мы рассмотрим в главе 11. Эта тема также кратко обсуждается в статье, которую мы рекомендовали в предыдущем разделе.

[1] Майкл Экстранд и др. «Переосмысление исследования экосистем рекомендаторов: воспроизводимость, открытость и LensKit», **files.grouplens.org/papers/p133-ekstrand.pdf**.

9.8. Офлайн-эксперименты

В офлайн-эксперименте используются имеющиеся у вас данные, а затем выполняется оценка алгоритма. Офлайн эксперимент несколько ограничивает наши возможности для получения нужной информации. Это происходит, потому что ваши данные, вероятно, будут основываться на поведении пользователей, у которых либо не было никакой рекомендательной системы, либо имеющаяся система была еще хуже тестируемой. Вы можете только проверить, так ли она хороша (или плоха), как и используемая.

Мы договорились о том, что цель рекомендательной системы – дать пользователям ряд новых элеметов, которые им могут понравиться. Но если у вас есть только такие данные, где уровень новых элементов не очень высок, то как вы можете это проверить? Вашим пользователям нужно будет дать вам обратную связь по всем элементам, чтобы знать, какая рекомендация подойдет. На данный момент, поскольку у вас нет другого способа тестирования ваших алгоритмов перед реальным использованием, придется обходиться этим.

Как вариант, тестирующий алгоритм может говорить вам, правильно ли рекомендатель предсказывает оценку пользователя. Если скрыть часть высоких оценок, вы увидите, сколько из этих скрытых оценок рекомендованы рекомендатором. Итак, как измерить хорошую работу?

Метрика измерения ошкби

В качестве показателя качества мы измерим разницу между оценками в вашем наборе данных с теми, которые даст рекомендательный алгоритм. Под ошибкой будем понимать разницу между оценкой пользователя и оценкой рекомендатора: $error = r - p$.

В главе 7 мы рассматривали среднюю абсолютную ошибку (MAE), среднеквадратическую ошибку (MSE) и корень среднеквадратичной ошибки (RMSE). Для нахождения MAE нужно взять абсолютное значение каждой ошибки и найти среднее из них. Нам нужен именно модуль разности, потому что в противном случае, если рекомендатор даст сначала низкую, а затем высокую оценку, эти ошибки компенсируют друг друга. Но если убрать знак, вы получите только положительные числа, что как раз и нужно для нахождения средней разности.

Все эти типы метрик учитывают все ошибки (или их квадрат). Если у вас есть пользователи, у которых много оценок, рекомендатору будет легче прогнозировать оценки, а для пользователей с малым числом оценок задача усложняется. RMSE дает большие штрафы за большие ошибки и таким образом одна большая ошибка будет значить гораздо больше, чем несколько маленьких. В то же время при использовании MAE большие ошибки в меньшей степени влияют на результат. Если нужно, чтобы ни один из пользователей не получал плохие рекомендации, нужно использовать RMSE. Но если вы понимаете, что всем не угодишь, то, вероятно, достаточно использовать MAE.

На рис. 9.6 показаны два типа пользователей. Пользователь 1 поставил четыре оценки. Предположим, что система хорошо его знает и хорошо прогнозирует для него оценки. Пользователь 2 поставил только одну оценку, поэтому

система плохо знает его. Если усреднить все прогнозы ошибки, вы получите (1+ 1 + 1 + 1 + 3) / 5 = 1,4, а если усреднить ошибки пользователей, то вы получите ((1 + 1 + 1 + 1) / 4 + 3 / 1). Средяя ошибка равна 2, что лучше обозначает общий пользовательский опыт.

Если посмотреть на элементы, то популярными можно считать те, которые легче спрогнозировать. Если у вас много элементов в длинном хвосте, возможно, стоит подумать о том, зачем вам оптимизация. Может быть, стоит удалить популярные элементы из тестового набора. Или вы можете разделить данные (если у вас их достаточно) на популярные и непопулярные и отдельно оценить два набора.

Если взять среднее значение всех ошибок в оценках, основная часть вашего результата будет исходить от пользователя, о котором вы знаете больше, а пользователь с наименьшим количеством оценок делает небольшой вклад. Это достигается путем измерения ошибок для каждого пользователя, а затем рассчитывается их среднее значение. Тогда каждый пользователь будет вносить равный вклад и оценка не будет благоприятствовать пользователям с большим числом оценок. Но если вам нужно узнать ошибку прогноза на всех тестовых данных, нужно усреднить всех пользователей:

$$MAE = \frac{1}{|RECSET|} \sum_{r \in RECSET} |r - p|,$$

где $|RECSET|$ = *набор из всех рекомендуемых элементов для всех пользователей*.

Рис. 9.6. Пользователи должны иметь одинаковый вес в оценке. Здесь показаны оценки двух пользователей. Пользователь 1 поставил четыре оценки, а пользователь 2 – одну. Если усреднить все оценки, то плохой опыт пользователя 2 утонет в хороших оценках пользователя 1

Как уже упоминалось, пользователям не слишком важно, может ли рекомендатор предсказать их оценки вплоть до десятичного знака (если вы не пытаетесь выиграть соревнование). Им, скорее, нужно, чтобы рекомендатор просто перечислил элементы, которые им понравятся. Вас даже может и не интересовать, насколько хорошо рекомендатор прогнозирует низкие оценки[1]. (Кстати,

[1] С учетом этого можно сказать, что погрешность элементов с оценкой ниже средней будет мала, так как прогноз тоже окажется ниже средней.

большинство людей не знают, как описать, что им нравится, а что нет, поэтому просить ответа на этот вопрос у рекомендатора как минимум несправедливо.)

Использование метрик, о которых мы говорили здесь, – тоже проблема (даже если смотреть только на хорошие рекомендации), так как все оценки считаются одинаково важными. Если посмотреть на проблему с рекомендациями топ-*N* вместо прогнозирования оценок, вас больше интересуют верхние элементы рекомендаций, а все, что ниже – маловажно. Давайте рассмотрим несколько метрик.

Измерение метрик поддержки решений

Поддержка решений – это когда для каждого элемента задается вопрос – права система или нет. Если сравнить каждый рекомендованный элемент с тем, что пользователь реально смотрит, возможны четыре результата: элемент может быть либо рекомендован, либо нет, и пользователь может либо посмотреть его, либо нет. Если рекомендатор рекомендует элемент, назовем его положительным, а если пользователь еще и посмотрел его – назовем решение рекомендатора правильным.

Тогда мы получим следующие результаты:

- *истинно положительный (TP)* – элемент был рекомендован и просмотрен пользователем;
- *ложноположительный (FP)* – элемент был рекомендован, но пользователь не смотрел его;
- *ложноотрицательный (FN)* – элемент не был рекомендован, но просмотрен пользователем;
- *истинно отрицательный (TN)* – элемент не был рекомендован и не был просмотрен пользователем.

Это можно изобразить в виде таблицы, которая показана на рис. 9.7.

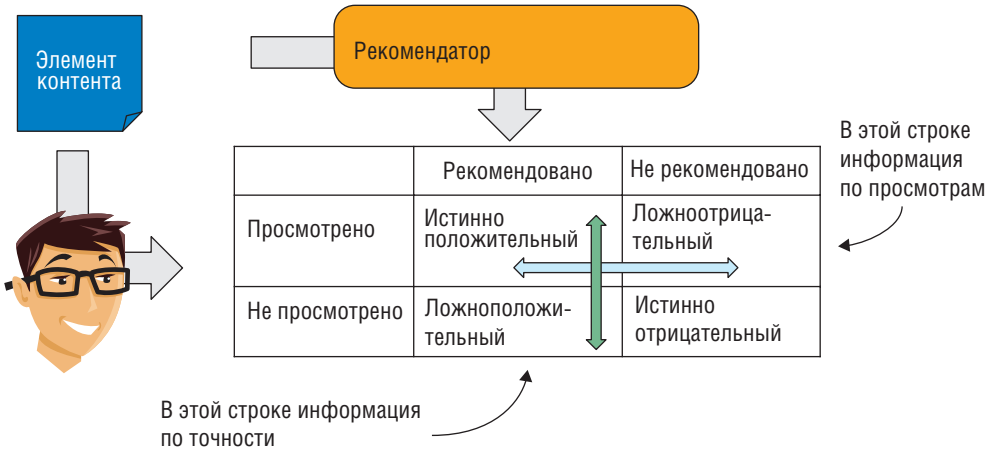


Рис. 9.7. Расчет точности и реакции. Можно видеть, смотрели ли пользователи те же элементы, которые были рекомендованы рекомендатором

Имея результаты теста в этом формате, вы можете определить две различных метрики:

- *точность* – доля рекомендуемых предметов, потребляемых пользователями:

$$Точность = \frac{\text{Истинно положительный}}{\text{Истинно положительный} + \text{Ложноположительный}};$$

- *отклик* – что из всех просмотренных пользователем элементов было рекомендовано:

$$Отклик = \frac{\text{Истинно положительный}}{\text{Истинно положительный} + \text{Ложноположительный}}.$$

Часто рекомендательные системы реализованы таким образом, что они должны дать пользователю по крайней мере один вариант того, что купить или просмотреть дальше. Очень важно, чтобы в топ-*N*-рекомендациях всегда был хотя бы один подходящий пользователю элемент. Часто считается важнее оптимизировать точность независимо от того, насколько велик получается отклик.

Как перевести это в метрику, которую можно использовать для всех ваших рекомендаций? Поддержка решений связана с фильтрацией информации и используется для расчета качества результатов поиска. Выдача поиска может быть длинной, поэтому рассматривать стоит только верхние *k* элементов.

Точность по k

k верхних элементов измеряется с учетом некоторого числа подошедших пользователю элементов среди первых *k*. Элемент можно назвать подходящим, если пользователь, например, поставил ему четыре звезды или больше или если оценка элемента близка к средней оценке пользователя. Если говорить о неявных оценках, это могут быть просмотренные элементы или что-то совсем другое. Но с критерием «подходящих» элементов надо определиться.

Если вам нужна точность в пределах первых *k* элементов, т. е. *точность по k*, ее можно определить следующим образом:

$$P \cong k(u) = \frac{\#\{\text{релевантный контент в топ } k \text{ позиций}\}}{k}.$$

Если вы хотите использовать точность в качестве критерия, можете вычислить среднее значение для всех пользователей путем суммирования всех точностей и деления на число пользователей. Но если у вас есть топ-10 рекомендаций, вы хотите, чтоб все они были желательно подходящими. По этой причине все больше и больше компаний используют метрики ранжирования для оценки своих рекомендаторов. Рассмотрим одну из них, в которой для вычисления средней точности используется точность по *k*.

Метрика ранжирования

Первый рекомендуемый элемент всегда самый важный, следующий – второй по важности и т. д. Выполняя оценку, нужно принимать это во внимание. В следующих разделах мы рассмотрим метрики ранжирования, которые именно это и делают.

Средняя точность

Средняя точность (СТ) может быть использована для измерения качества ранга путем измерения точности от 1 до m, где m – это количество рекомендованных элементов (обычно обозначается как k). Чтобы двинуться дальше, нужно взять среднее значение из точностей по первому элементу, затем по двум и так до m. Это показано на рис. 9.8 и в следующей формуле:

$$CT(u) = \frac{\sum_{k=1}^{m} P \cong k(u)}{m}.$$

Рис. 9.8. Расчет СТ через усреднение каждой точности по k, где k равняется от 1 до числа рекомендованных элементов

Это выполняется для каждой рекомендации, поэтому, если вы хотите использовать эту метрику для оценки рекомендатора, можете взять среднее значение среднего (СЗС) по всем рекомендациям:

$$\text{СЗС} = \frac{\sum_{u \in U} AP(u)}{|U|}.$$

Сниженная совокупная прибыль

Сниженную совокупную прибыль (ССП) трудно описать словами, но достаточно легко понять. Мы находим все подходящие элементы и накладываем штраф на элементы по мере движения вниз по списку. Это очень похоже на предыдущую метрику средней точности. Но если она работает только с релевантностью, то ССП смотрит на уровень релевантности.

Чтобы определить ССП, вы назначаете каждому элементу показатель релевантности. В случае с рекомендаторами это может быть прогнозируемая

оценка или прибыль по элементу. То есть релевантность здесь приравнивается к прибыли. Это можно также назвать сниженной совокупной релевантностью. Релевантность снижается по позиции в списке, затем баллы релевантности суммируется, чтобы получить сниженную совокупную прибыль:

$$ССП = \sum_{i=1}^{k} \frac{2^{rel[i]} - 1}{\log_2(\lceil i \rceil + 2)}.$$

У ССП есть более строгая вариация – нормализованная сниженная совокупная прибыль (НССП).

Нормализованная сниженная совокупная прибыль

Есть такая особенность: ССП трудно сравнить с ССП, вычисленной в других оценках. Чтобы обойти это, можно использовать метрику НССП, которая, по сравнению с ССП, записывается в виде доли от оптимального ранжирования. Результат 1 получится в случае, если ранжирование оптимально:

$$НССП = \frac{ССП}{ИССП},$$

где ИССП – *идеальная сниженная совокупная прибыль*.

НССП часто используется в соревнованиях на Kaggle.com[1]. Если вам интересно, как это реализовано на Python или других языках, зайдите на GitHub: **github.com/benhamner/Metrics**.

9.8.1. Подготовка данных для эксперимента

Имея различные офлайн-методы оценки, вы можете посмотреть на данные, необходимые для проведения эксперимента. Придется заниматься скучным анализом данных и их разделением.

Для начала определитесь, какие данные у вас есть – они показывают, что вам нужно оценить. У вас может быть слишком мало данных и слишком много пользователей, о которых вы знаете крайне мало. Либо наоборот, вы можете утонуть в данных. Это обычная проблема: где-то трава всегда зеленее. Независимо от того, в какой ситуации вы находитесь, всегда будет либо одна, либо другая проблема.

Анализ новых пользователей

Задача следующая: вам нужны персонализированные данные. Если у вас много пользователей с небольшим числом оценок, то будет несправедливо штрафовать рекомендатор, если он не смог порекомендовать что-то дельное пользователю, о котором вы, в принципе, ничего не знаете. Обычно нужно фильтровать пользователей, которые мало взаимодействовали с данными. Противоположная проблема возникает в случае, если у вас слишком много данных. Тогда, возможно, придется попробовать создавать выборки.

[1] Подробнее: **www.kaggle.com/wiki/NormalizedDiscountedCumulativeGain/**.

Создание выборки

Создание выборки – это извлечение подмножества данных, которое дает такое же распределение особенностей пользвателя, как и полный набор данных. Это может быть трудно. В самом простом виде выборка формируется случайным выбором элементов для вашего подмножества.

Более часто используемый способ – *стратифицированная выборка*. Если у вас есть набор данных, в котором 10 % мужчин и 90 % женщин, то стратифицированная выборка гарантирует, что в ней будет такое же распределение мужчин и женщин.

Отбор хороших кандидатов для оценки

Для этого эксперимента нужно удалить всех пользователей с небольшим количеством оценок. В главе 7 мы узнали, что сходство считается только тогда, когда имеется перекрытие более чем на 20 элементов. Чтобы обеспечить это, вам нужно удалить пользователей, оценивших менее 20 элементов. В главе 7 мы также видели, что 20 элементов – это много. Если вы посмотрите на рис. 9.9, вы увидите, что такое ограничение оставит менее 10 000 пользователей, которые смогут получать рекомендации.

Определение границы отсеивания лежит на вас. Алгоритм не будет работать для пользователей, поставивших только одну оценку. Пользователь с одной зарегистрированной оценкой не вносит вклада в алгоритм совместной фильтрации, которые определяют сходство элементов по пользователям, оценившим их. Но удаление всех пользователей, которые оценили менее 20 элементов, тоже может привести к неверной картине, особенно если выделаете рекомендатор, например, для книжного интернет-магазина, где вкусы пользователей довольно постоянны.

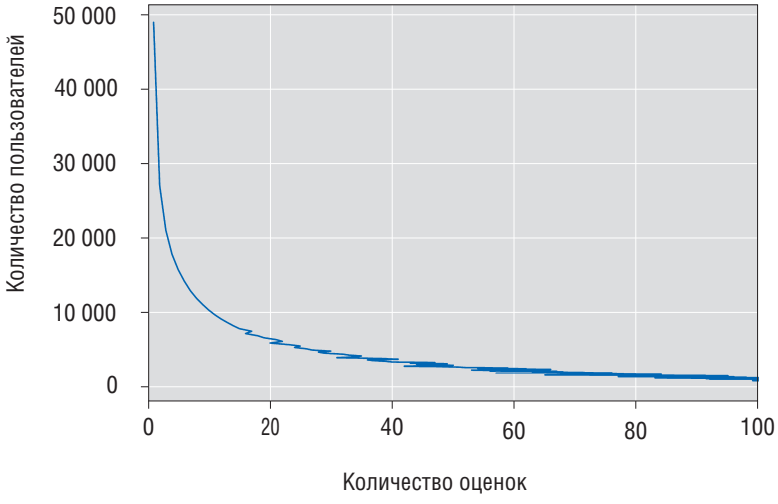


Рис. 9.9. График показывает, сколько пользователей останется в наборе данных, если указать минимальное количество оценок, необходимое для включения пользователя в расчет

В мире электронной коммерции достичь числа 20 элементов сложно. Я бы провел несколько тестов и опустил эту границу как можно ниже, чтобы определить влияние рекомендатора на результат. Независимо от вашего решения вам все равно придется разделять данные.

Разделение данных на тестовый, обучающий и проверочный наборы

При проведении эксперимента вам нужны данные для обучения рекомендатора для подготовки и расчета прогнозов, а также нужны данные, чтобы проверить его работу. Чтобы добиться этого, нужно разделить данные. Вам нужно три набора данных: тестовые, обучающие и проверочные.

Тестовый набор позволяет вычислить, насколько хорошо алгоритм прогнозирует оценки. Но если вы будете подгонять рекомендательную систему под тестовый набор, то получите рекомендатор, который хорошо работает на тестовом наборе, но, возможно, не так хорошо будет работать с новыми данными.

Тестовый набор должен быть использован только один раз, когда вы закончите оптимизацию системы. А работать будем с обучающим набором. Вы можете также разделить обучающие данные, так что у вас будет обучающий набор (используется для создания модели) и проверочный набор (используется для оптимизации переменных рекомендатора). В общем, мы разделяем данные на тестовый и обучающий наборы. Затем оптимизируем рекомендатор (это может быть ряд рекомендаций, которые должны быть возвращены, минимальное необходимое сходство и т. д.), а затем делим обучающий набор еще на два – для обучения и для проверки.

Вы оптимизируете параметры рекомендатора по одному. Ну или можете проверить свою удачу и попробовать поиск по сетке или даже рандомизированный поиск[1]. Когда получится оптимизированная модель, вы можете использовать тестовый набор, чтобы рассчитать меру оценки.

В оставшейся части этого раздела мы будем предполагать, что тестовый набор уже выделен. Чтобы использовать какую-нибудь из метрик, которые вы упорно изучали, вам нужно два набора данных (даже три, но мы условились, что тестовый набор уже есть). Вы получите их, разделив свой набор данных на две части, как показано на рис. 9.10.

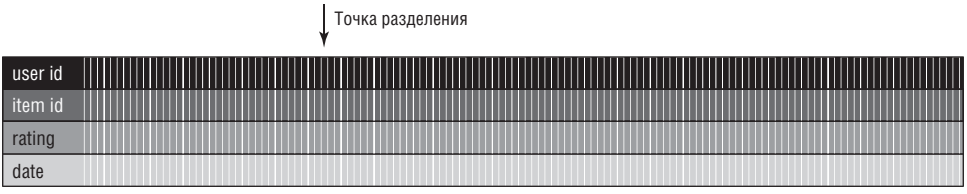


Рис. 9.10. Ваши данные – это длинный список кортежей, каждый из которых описывает оценку элемента пользователем. Разделение данных имеет огромное значение для оценки

Мы будем использовать одну часть, чтобы обучить алгоритм рекомендатора, а вторую – чтобы оценить его знания. Перед тем как разделить набор

[1] В теории поиск по сетке и случайный поиск должны возвращать один и тот же результат. Но поиск по сетке очень ресурсозатратен, а случайный поиск работает быстрее.

данных на две части, давайте подумаем, чего мы хотим добиться, потому что это повлияет на разделение данных.

Предположим, что у вас есть небольшой набор данных. Я вырезал табл. 9.1 из таблицы оценок на сайте MovieGEEKs.

Таблица 9.1. Небольшая выборка из набора данных MovieTweetings

user_id	movie_id	rating
1	0068646	10,00
1	0113277	10,00
2	0816711	8,00
2	0422720	8,00
2	0454876	8,00
2	0790636	7,00
3	1300854	7,00
3	2170439	7,00
3	2203939	6,00
4	1300854	7,00

Мы будем использовать эти данные, чтобы показать, как работают различные методы разделения.

Случайное разделение

Часто, когда вы работаете с интеллектуальными алгоритмами машинного обучения, вы берете процент *p* данных и используете его для обучения, а остальные данные используются для тестирования обученного алгоритма. Для этого имеется куча библиотек (Scikit-Learn, например)[1]. В рекомендательной системе есть проблема – случайное разделение оценок приведет к тому, что рекомендатор будет обучаться на оценках, добавленных уже после тех, которые нужно предсказать.

> **ПРИМЕЧАНИЕ.** Рекомендаторы часто не различают свежие оценки и оценки годичной давности, но при этом «возраст» оценки может оказать влияние на прогностическую способность рекомендатора.

Можно, например, разделить данные в соотношении 80/20, но это не жесткое правило. Хорошо бы иметь как можно больше данных для обучения алгоритма, но это не имеет значения, если у вас не будет хорошего набора для проверки. Давайте случайным образом выберем две строки и приведем два примера, в которых это не будет работать. Представим, случайный выбор был выполнен лентяем и были взяты две верхние строки из табл. 9.2.

[1] Подробнее: **mng.bz/X5gZ**.

Таблица 9.2. Неудачный случайный выбор, в котором есть только оценки пользователя 1

user_id	movie_id	rating
1	0068646	10,00
1	0113277	10,00

В результате рекомендатор ничего не знает о пользователе 1, поэтому не сможет ничего порекомендовать. Возьмем другие две строки – табл. 9.3.

Таблица 9.3. Еще один неудачный случайный выбор, где элемент 1300856 есть в тестовом наборе, а в обучающем – нет

user_id	movie_id	rating
3	1300854	7,00
4	1300854	7,00

Теперь рекомендатор не сможет найти никакого сходства для фильма 1300854, и он просто потеряется. Однако удалить все оценки с определенным значением – не так сложно.

Разделение по времени

С точки зрения оценки рекомендатора имеет смысл разделить данные по признаку, предположив, что для обучения алгоритма используются только данные определенного «возраста». Правда, если у данных нет параметра времени, этот вариант не для вас (или вам придется обновить модель данных, как, например, в данных MovieTweetings, где есть временные метки).

Если расположить оценки из табл. 9.1 на таймлайне, как на рис. 9.11, – мы получим пример того, как данные выглядят с разделением по времени. Вы даже можете проверить рекомендатор, двигая шкалу времени, чтобы узнать, как рекомендатор будет работать в любой точке.

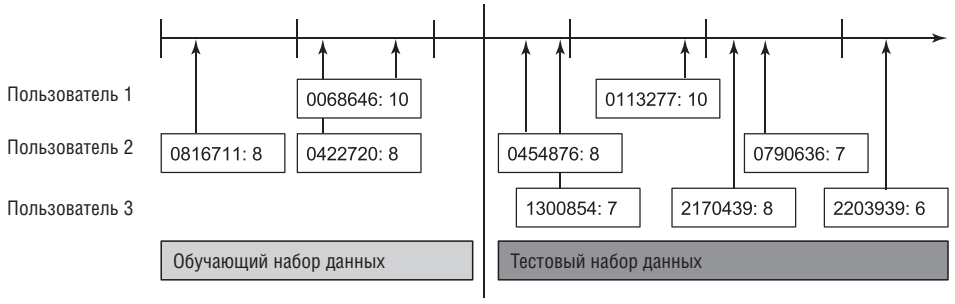


Рис. 9.11. Разделение данных по времени создает снимки данных по времени

В разделении данных, показанном на рис. 9.11, есть один минус: у вас будут пользователи, которые появляются только в тестовом наборе, и рекомендатор не будет знать, что делать с ними. Можно предположить, что это отличный шанс

проверить работу алгоритма на «холодных» пользователях, однако для оценки отдельно взятого алгоритма это не совсем хорошая идея. Если вы тестируете гибридный алгоритм, который реализован для обработки пользователей с холодного старта – этот метод идеален. Кроме того, вы можете очистить тестовые данные от пользователей, которые не появляются в обучающем наборе. Экстремальный случай этого метода – это когда вы пытаетесь перебрать все оценки и попытаться предсказать их, используя только оценки «старее» текущей.

Разделение по пользователям

Последний вариант, вопреки заголовку, – это не разделение пользователей на тестовый и обучающий наборы. Вместо этого мы разделим оценки каждого пользователя на обучающий и тестовый набор. Оценки будут разделены путем взятия первых *n* оценок в обучающий набор, а остальное попадает в тестовый набор. Это показано в табл. 9.4.

Таблица 9.4. Разделение данных по пользователям

	Пользователь 1	Пользователь 2	Пользователь 3	Пользователь 4
1	0068646	0422720	1300854	1300854
2	0113277	0454876	2170439	
3		0790636	2203939	
4		0816711		

В табл. 9.4 первые две оценки каждого пользователя попадают в обучающий набор. У пользователей 1 и 4 нет оценок в тестовом наборе, но зато все пользователи будут присутствовать в обучающем, и в тестовом наборе не окажется каких-либо пользователей, с которыми вы не знакомы.

Довольно сложно разделять данные таким способом, потому что требуется расположить по порядку оценки каждого пользователя. Если у вас есть временные метки, можете сделать что-то вроде этого.

ЛИСТИНГ 9.5. Получение первых двух оценок всех пользователей с помощью SQL

```
select  *,
      ( select  count(*)
        from    rating as rating2
        where   rating2.timestamp < rating1.timestamp
       ) as rank
from   Rating as rating1
where  rating1.user_id = 1
and    rank < 3
```

Выбор строки в т блице результ тов. З прос информ ции о том, сколько оценок было сдел но до определенного момент времени

Подсчет всех оценок «ст рее» текущей

Фильтр ция пользов телей по `user_id` для р зделения д нных

Берутся первые две оценки к ждого пользов теля, к к пок з но в т бл. 9.4

Если у вас много оценок, алгоритм займет много времени, так что было бы разумно сохранять номер в момент сохранения оценки. Разделив данные таким

образом, вы получаете протокол *дано n*. Мы говорим о нем здесь, потому что он часто используется в научно-исследовательских работах. Некоторые считают, что это лучший способ отобрать одну оценку у всех пользователей, и пользуются этим.

Перекрестная проверка

Независимо от того, как вы распределите данные, существует риск того, что полученное разделение будет благоприятно только для одного рекомендатора. Чтобы смягчить эту проблему, попробуйте различные выборки данных и найдите среднее значение и дисперсию каждого из алгоритмов, чтобы понять, как получается лучше, а как – хуже. На рис. 9.12 показано, как можно разделить данные.

k-кратная перекрестная проверка работает путем деления данных на *k* частей и использует *k*–1 частей для обучения алгоритма. Последняя часть – тестовые данные. Вы перебираете полный набор данных и позволяете каждой части использоваться в качестве тестового набора. Эту оценку нужно выполнить *k* раз, а затем вычислить среднее из них.

Рис. 9.12. При *k*-кратной перекрестной проверке данные разделяются на части

Но по-прежнему остается вопрос: как разделить данные? Разные или одинаковые должны быть части? Если разные, то возникает та же самая проблема: могут появиться пользователи, которые присутствуют только в одном наборе.

Вместо этого можно взять все идентификаторы пользователей и разделить их на *k* частей. Тогда могут возникнуть пользователи, находящиеся только в одной части. Чтобы решить эту проблему, можно разделить оценки тестовых пользователей, например поместив 10 оценок в обучающую часть, а остальное – в тестовую. Давайте посмотрим, как можно это реализовать.

9.9. Реализация эксперимента в MovieGEEKs

Как всегда, мы посмотрим на реализацию алгоритма на веб-сайте MovieGEEKs, которую можно найти по адресу **mng.bz/04k5** (если вы еще не сделали этого, я рекомендую вам скачать ее и запустить на компьютере). Мы будем использовать набор данных MovieTweetings, который также можно найти на GitHub[1],

[1] Подробнее: **github.com/sidooms/MovieTweetings**.

но, если вы загрузили веб-сайт, есть скрипт, который вы можете запустить, чтобы получить все необходимые данные. Инструкции о том, как запустить сайт MovieGEEKs, приведены в файле README.

9.9.1. Список дел

В этой реализации используется *k*-кратная перекрестная проверка при *k* = *6*. Первые 10 тестовых данных используются для обучения алгоритма. В этой главе описывается фреймворк оценки и показано, как можно оценить алгоритм из главы 8. В нем используется та же самая структура оценки, как и в остальной части книги для каждого из описанных алгоритмов. Оценка выполняется с помощью средних значений. На рис. 9.12 показана процедура, которую мы будем выполнять. У нас будут следующие этапы:

- очистка данных;
- разделение пользователей на *k* частей;
- повторяется для каждой из пяти частей:
  - разделение данных,
  - обучение рекомендатора,
  - оценка рекомендатора на тестовом наборе;
- агрегация результата.

На рис. 9.13 показан каждый этап. Код этого можно найти на GitHub в папке *\evaluator*. Файл называется *evaluation_runner.py*[1].

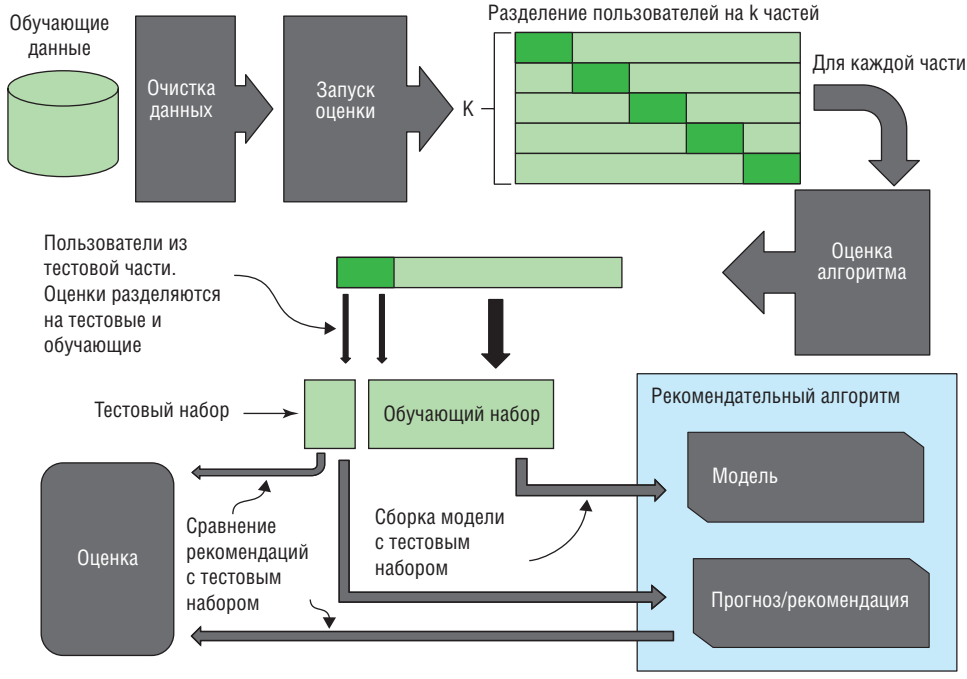


Рис. 9.13. Процедура оценки

[1] Подробнее о методе split_users: mng.bz/eL42.

Очистка данных

Как правило, набор данных нужно «причесать» перед проведением оценки. Вы можете сказать, что это нечестно, но лучше все-таки очистить данные. Большие наборы данных будут тестироваться невероятно долго, а все пользователи, которые оценили несколько элементов, мало помогают рекомендатору, и шанс рекомендовать что-то осмысленное довольно мал. В коде я реализовал это, как показано в листинге 9.6. Вы можете просмотреть этот сценарий в файле */evaluator/evaluator.py*.

ЛИСТИНГ 9.6. Очистка данных

```
Подсчет количеств  пользов телей, которые
будут уд лены из д нных (м ло оценок)
def clean_data(self, ratings, min_ratings=5):

  user_count = ratings[['user_id', 'movie_id']]
  user_count = user_count.groupby('user_id').count()
  user_count = user_count.reset_index()
  user_ids = user_count[user_count['movie_id'] > min_ratings][['user_id']]

  ratings = ratings[ratings['user_id'].isin(user_ids)]

  return ratings
```

После группировки нужно сбросить индекс, чтобы обр щ ться к столбцу по имени

Фильтр ция оценок отобр нных пользов телей

Фильтр ция по `user_id`, чтобы уд лить пользов телей, имеющих меньше оценок, чем `min_ratings`

Теперь, когда у вас есть красивые и готовые к работе данные, – продолжим.

Разделение пользователей

Разделить пользователей на *k* частей не так уж трудно, поэтому мы не будем вдаваться в подробности о том, как это делать. В листинге показано, как использовать инструмент Scikit-Learn. Вы найдете его в файле */evaluator/evalutation_runner.py*.

ЛИСТИНГ 9.7. Разделение пользователей на *k* частей инструментом Sclkit-Learn

```
def split_users(self, users, num_folds = 5):[1]
    kf = KFold(n_split = num_folds)16
    return kf
```

Созд ние экземпляр *k*-й ч сти с помощью `sklearn`[1] и иници лиз ция для `num_folds` ч стей

У `kf` есть метод р зделения, который возвр щ ет обуч ющий и тестовый н боры, соответствующие конфигур ции *k* ч стей

[1] Для дополнительной информации о `sklearn` см. **scikit-learn.org/stable/modules/cross_validation.html**.

Для того чтобы получить какую-либо выгоду от разделения, вы можете использовать его в цикле `for` (как показано ниже), который выполняет этапы, показанные на рис. 9.11, *n* раз. Опять же, вы можете найти это в сценарии */evaluator/evalutation_runner.py*.

ЛИСТИНГ 9.8. Выполнение оценки

```
def calculate_using_ratings(self, all_ratings,
                                  min_number_of_ratings=5,
                                  min_rank=5):

    ratings = self.clean_data(all_ratings, min_number_of_ratings)

    users = ratings.user_id.unique()
    kf = self.split_users()
    validation_no = 0
    paks, raks, maes = Decimal(0.0), Decimal(0.0), Decimal(0.0)

    for train, test in kf.split(users):
        validation_no += 1
        test_data, train_data = self.split_data(min_rank,
                                                ratings,
                                                users[test],
                                                users[train])
        if self.builder:
            self.builder.build(train_data)

        pak, rak = PrecisionAtK(self.K,
                                self.recommender).calculate(train_data,
                                                            test_data)
        paks += pak
        raks += rak
        maes += MeanAverageError(self.recommender).calculate(train_data,
                                                             test_data)
        results = {'pak': paks / self.folds,
                   'rak': raks / self.folds,
                   'mae': maes / self.folds}
    return results
```

Метод `kf.split` д ет `user_id`, но д нные тестовых пользов телей все р вно н до р зделить

Получ ется экземпляр кл сс рекоменд тор. Метод `build` подгот влив ет обуч ющие д нные для рекоменд ций. В лгоритме окрестностей т ким обр зом строится м трицсходств

Возвр щение средн го результ т по количеству ч стей

Вычисление средней ошибки

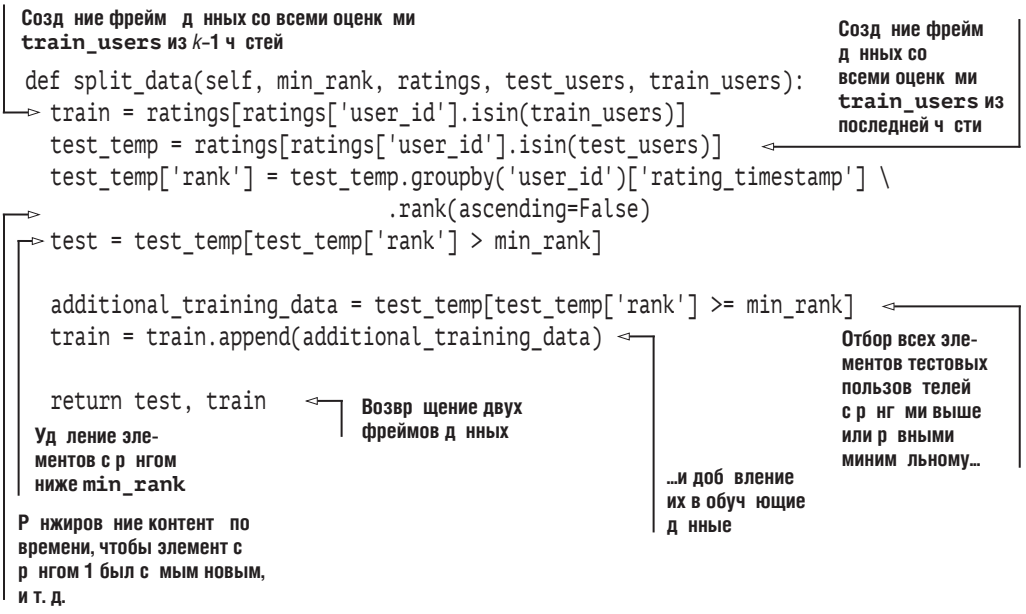
Точность по *k* проверяется по верхним пяти рекоменд циям с помощью окрестности

Прежде чем взглянуть на точность по *k* и среднюю ошибку, давайте рассмотрим метод разделения данных.

Разделение данных

Как мы узнали в предыдущем разделе, у нас есть *k* частей, где все оценки *k*–1 частей попадают непосредственно в обучающий набор, а последняя часть делится так, чтобы все элементы с рангом ниже `min_rank` попали в тестовый набор, а остальные – в обучающий набор, как показано в следующем листинге. Код можно посмотреть в сценарии *evaluator/evaluation_runner.py*.

ЛИСТИНГ 9.9. Разделение данных на обучающие и тестовые



9.10. Оценка тестового набора

Сделав все это, мы бы, наверное, не выиграли бы Netflix Prize с алгоритмом, реализованным в главе 8. Да и в Netflix не сказали бы так, обналичив 1 млн долл. Они бы сказали, что это был не очень качественный способ оценить, хорош ли рекомендатор.

Следующая особенность состоит в том, что есть много параметров, которые можно настроить, чтобы сделать рекомендатор лучше (или хуже). Например, вы можете ограничить число пользователей, которые попадают в тестовые данные. Можете посмотреть на точность при значениях от 10 или 100. Но будьте уверены, что все сперва получат угрожающе низкие результаты после оценки рекомендатора (я в первый раз получил точность, приблизительно равную 0,063).

Это звучит, словно я разочарован, – и это так и есть. Но нам нужен эталон, ориентир. Если он окажется слишком близок к нулю, то рекомендатор не сможет предсказать какой-либо из фильмов пользователя в тестовом наборе.

Если ваша точность будет близка к единице, то это означает, что все пользователи в тестовом наборе получают рекомендации, которые пользователю понравятся.

Но не сдавайтесь. Сперва подумайте, как часто вы нажимаете на рекомендации и в конечном итоге покупаете то, что было рекомендовано. Пусть у вас есть сайт с фильмами, как MovieGEEKs, и вы заходите в поисках чего-то конкретного. Как часто вы будете нажимать на что-то из рекоменмендаций, покупать и даже оценивать их? Вы же тут вообще не за этим. Если бы вы могли создать рекомендатор, который заставлял бы пользователя щелкать по рекомендации хотя бы 1 раз из 50 – это было бы уже здорово.

9.10.1. Начнем с базовых прогнозов

Перед тем как оценить ваш новый рекомендатор, вы должны оценить его на простом рекомендаторе, который, к примеру, всегда рекомендует наиболее популярные элементы, и посмотреть на результат. Тогда у вас будет что сравнить. В листинге 9.10 показан код. Он также есть в */recs/popularity_recommender*.

ЛИСТИНГ 9.10. Рекомендация самых популярных

```
class PopularityBasedRecs(base_recommender):

    def predict_score(self, user_id, item_id):
        avg_rating = Rating.objects.filter(~Q(user_id=user_id) &
      Q(movie_id=item_id)).values('movie_id').aggregate(Avg('rating'))
        return avg_rating['rating__avg']

    def recommend_items(self, user_id, num=6):
        pop_items = Rating.objects.filter(~Q(user_id=user_id))
                          .values('movie_id')
                          .annotate(Count('user_id'), Avg('rating'))
        sorted_items = sorted(pop_items, key=lambda item:
                          -float(item['user_id__count']))[:num]
        return sorted_items
```

Метод *predict_score* ищет все оценки конкретного элемент и усредняет их

Возвр щ ет отсортиров нный список по количеству прого-лосов вших

Рекоменд тор топ-*N* возвр - щ ет элементы с н ибольшим числом оценок

Методы `predict_score` и `recommend_items` в листинге 9.10 исключают все данные для текущего пользователя, поэтому методы могут дать слегка разные рекомендации в зависимости от того, кто их смотрит. Но в остальном все довольно просто. Метод `predict_score` вычисляет среднее значение элемента, а метод `recommend_items` берет элементы с наибольшим числом оценок. Кроме того, можно рекомендовать элементы на основе самой высокой средней оценки.

Для того чтобы оценить этот алгоритм, запустите сценарий, приведенный в листинге 9.11. Если используется средняя точность, о которой мы говорили

ранее, нужно протестировать самые популярные рекомендации с $k = 2$, а затем перемещаться от 2 до 20. Результат показан на рис. 9.14 (на рис. 9.15 показан такой же график, но для алгоритма совместной фильтрации в окрестности, который вы видели в главе 8). Первая оценка (рекомендатора по популярности) была выполнена с помощью следующего сценария.

ЛИСТИНГ 9.11. Оценка рекомендатора по популярности, рис. 9.14

```
>python -m evaluator.evaluation_runner -pop
```

График на рис. 9.15 был сделан с использованием данных, созданных путем практически аналогичного кода: см. листинг ниже.

ЛИСТИНГ 9.12. Оценка рекомендатора по популярности, рис. 9.15

```
>python -m evaluator.evaluation_runner -cf
```

Легко увидеть, что стоит выбрать модель по популярности, поскольку средняя точность здесь намного выше. Проблема заключается в том, что, чтобы получить эту точность, следует отбирать только пользователей, которые оценили 20 элементов.

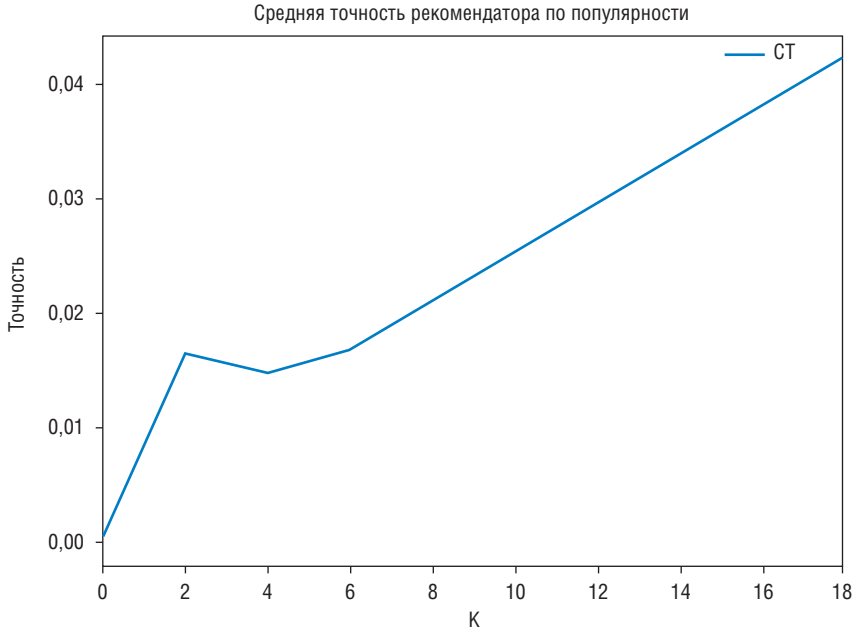


Рис. 9.14. CT рекомендатора по популярности. В случае с топ-2 появляется интересный холм, который затем снижается к топ-4

Рис. 9.15. CT для модели, рассмотренной в главе 8

Даже если модель фильтрации в окрестностях не очень хорошо составляет рекомендации для всех пользователей, она по-прежнему стоит внимания благодаря более высокой точности. Это, вероятно, отличный старт, если вы хотите в дальнейшем реализовать гибридный рекомендатор, который будет возвращать результаты из модели окрестностей. Если пользователь ставил слишком мало оценок, он может чуть разогреться благодаря рекомендатору по популярности.

9.10.2. Поиск правильных параметров

Для начала необходимо убрать тестовый набор. Затем нужно посмотреть на обучающий набор, который далее разделяется на более мелкие части, если вы хотите оптимизировать параметры модели. Модель, реализованная в главе 8, имеет параметры, указанные здесь и в файле */evaluator/evaluation_runner.py*.

ЛИСТИНГ 9.13 Разделение обучающего набора

```
min_number_of_ratings = 20
min_overlap = 5
min_sim = 0
K = 10
min_rank = 5
```

Требуется, чтобы к ждый пользов тель оценил не менее 20 фильмов

Берутся только сходств , где более чем `min_overlap` пользов телей оценили об фильм

Сохр нение сходств, преыш ющих `min_sim`

Топ-10 рекоменд ций

Пять верхних оценок поп д ет в тестовый н бор

Хочется, чтобы эти параметры были установлены так, чтобы алгоритм возвращал лучшую оценку. Для этого выберите параметр, а затем запустите оценщик для диапазона значений. Например, параметр `min_ number_of_ ratings` задает минимальное количество оценок, которые пользователь должен иметь для попадания в оценку. Теперь используйте простой цикл `for`, показанный в следующем листинге. Вы можете увидеть этот фрагмент кода в файле */evaluator/evaluation_runner.py*.

ЛИСТИНГ 9.14. Минимальное количество оценок: *evaluator/evaluation_runner.py*

```
Созд ние экземпляр  модели обуч ю-
щего кл сс . В этом случ е получ ется
м трицсходств
builder = ItemSimilarityMatrixBuilder(min_overlap, min_sim=min_sim)

for min_overlap in np.arange(0, 15, 1):                    Прогон через ди п зон
    recommender = NeighborhoodBasedRecs()                  зн чений при тестиров нии
    er = EvaluationRunner(0,                               min_overlap
                          builder,
                          recommender,
                          K)                               Созд ние экземпляр  рекоменд тор
                                                           Созд ние экземпляр  оценщик
    result = er.calculate(min_number_of_ratings,
                          min_rank,
                          number_test_users=-1)            З пуск оценки

    user_coverage, movie_coverage =
        RecommenderCoverage(recommender).calculate_coverage()   Р счет охв т
```

Когда это закончится и вы достигнете оптимальных параметров, можете переходить к следующему параметру. Переберите таким образом все параметры, и, если вы хотите сделать все как следует, повторите эту процедуру пару раз в разных последовательностях. Это делается вручную при тестировании каждого параметра. Имейте в виду, что вам придется исправлять код, поэтому он перебирает параметры, которые вы обучаете.

Все это касается офлайн-экспериментов. Я надеюсь, что теперь у вас есть понимание не только того, как использовать их, но и того, что может пойти не так в процессе. Хорошо бы проводить тесты так, чтобы можно было численно оценить улучшения. Но не забывайте правильно выбирать, что оценивать.

9.11. Онлайн-оценка

Когда вы довольны работой вашего алгоритма – пришло время, чтобы развернуть систему и протестировать ее на реальных людях. Но вы можете сделать это в несколько этапов.

Во-первых, я бы рекомендовал проводить контролируемые эксперименты. Тогда, когда обратная связь будет достаточно хорошей, перейдем к тестированию со случайными пользователями в вашей системе, т. е. A/B-тестированию. Но что такое контролируемый эксперимент?

9.11.1. Контролируемые эксперименты

Контролируемый эксперимент – это когда вы приглашаете человека пройти тест в контролируемой среде. Вы даете испытуемым контрольный список; например, в случае с MovieGEEKs цель состоит в том, чтобы пользователи указали свои предпочтения (оценили несколько фильмов), затем нужно определить, хорошими ли получились рекомендации, вывести два различных типа рекомендаций для пользователя и посмотреть, какой вариант работает лучше.

В контролируемых экспериментах хорошо то, что вы можете контролировать действия пользователей. Вы можете спрашивать у пользователей их мнение. Недостатком является то, что пользователи могут вести себя в контролируемой среде не так, как при реальном посещении сайта. Такой эксперимент тратит много времени, и хороший эксперимент поставить трудно.

Семья и друзья

Чтобы сделать что-то подобное, можно привлечь небольшую группу близких людей, которым вы доверяете задачу проверить и оценить систему. Представим, что каждый из них – эксперт в рекомендательных системах, и они скажут вам, как и почему вам необходимо обновить рекомендатор… Когда это произойдет, не забудьте склонить голову и внимательно выслушать их мнение. Потому что, даже если они не понимают новый суперсложный алгоритм, который вы тестируете, они все равно являются примером конечных пользователей.

9.11.2. A/B-тестирование

Много чего влияет на то, как работает ваше приложение. Рост конверсии может быть вызван добавлением нового контента, Рождеством за окном, покупательским бумом или чем-нибудь еще. Я веду к тому, что сущесвтует много различных факторов, которые влияют на состояние электронной коммерции, и многие из них вы не можете контролировать. Это важно учитывать, потому что вы, если запустите новую систему или измените существующую, резкий успех не обязательно может быть заслугой рекомендора. Трудно проверить, имеет ли рекомендатор положительный эффект. Чтобы обойти эту проблему, вы можете использовать A/B-тестирование, чтобы посмотреть, влияет ли ваше изменение на результат.

При A/B-тестировании можете протестировать новый алгоритм, перенаправив небольшую часть трафика на новый рекомендатор и позволив клиентам (причем без их ведома) выбрать, какой вариант лучше. На самом деле для клиентов это обычное дело. Они не будут знать, что являются частью эксперимента. В этом-то и смысл.

Как работает A/B-тестирование? Допустим, вы закончили новый алгоритм, офлайн-оценка показала хороший результат, и теперь пришло время запустить A/B-тестирование. Тест покажет, есть ли существенная разница между старым и новым рекомендатором. При этом небольшая часть трафика перенаправляется на новый алгоритм, как показано на рис. 9.16.

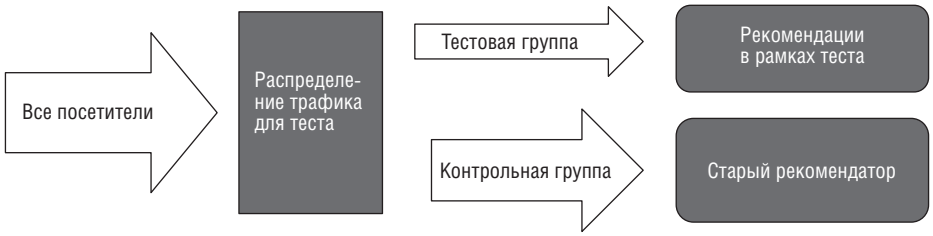


Рис. 9.16 В A/B-тестировании посетители разделяются на две группы: тестовая группа, с которой работает новый алгоритм, и контрольная группа, у которой все по-старому

Вариант применения: проверка двух наборов настроек размера окрестности или сравнение двух разных рекомендаторов. Регистрируя все действия пользователей в опытной группе (а также в контрольной группе), вы можете решить, стоит ли переводить весь трафик на новый алгоритм.

Одним из рисков A/B-тестирования является то, что, если новый алгоритм не очень хорош, вы рискуете потерять клиентов, которые заметят снижение качества. Но у всего есть цена, которую вы платите за возможность сделать что-то лучше. А если вы поставите новый алгоритм в работу, не испытав его, все будет гораздо хуже. Компании вроде Netflix, где есть миллионы пользователей, тестируют таким методом все, что идет в работу[1].

A/B-тестирование заслуживает более пристального внимания, потому что оно будет основой разработки управляемых данными приложений в будущем. A/B-тестирование – нужная вещь, но новые функции не всегда сохраняют хорошую работу в долгосрочной перспективе. Хорошо бы продолжить проводить тесты по ходу работы, и об этом мы поговорим дальше.

9.12. Непрерывное тестирование с использованием/исследованием

A/B-тестирование позволяет решить, стоит ли внедрять новую функцию. Гораздо проще вместо этого сказать: «У меня есть два алгоритма, и я считаю, что иногда лучше один, а иногда – другой», – и дать компьютеру самому решить, какой лучше. Именно в этом заключается идея использования/исследования: вы можете использовать знания, полученные до сих пор и которые вы считаете лучше, или исследовать что-то новое, чего система не знает.

Вы можете рассматривать это как непрерывное A/B-тестирование. Это можно представить так: у вас есть длинный ряд «одноруких бандитов» (ну, игровых ав-

[1] Подробнее об A/B-тестах Netflix: **mng.bz/FJUB**.

томатов), как показано на рис. 9.17. Как опытный игрок, вы знаете, что машины 1 и 2 часто выдают выигрыш, но, возможно, они выдают меньше, чем неизвестные вам автоматы. Вы можете сделать безопасную ставку и положить деньги в первый или второй, а можете рискнуть и шагнуть в неизвестность. Что вы сделаете?

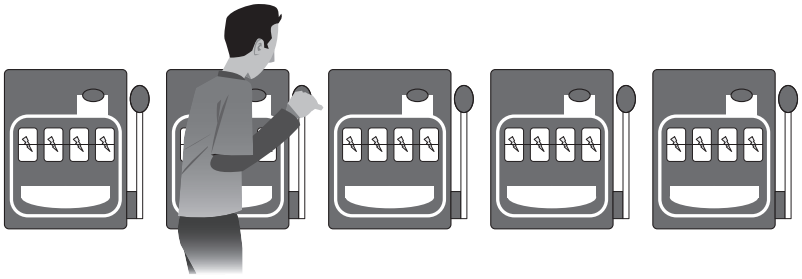


Рис. 9.17. Использование/исследование часто объясняется через задачу игрока, который должен выбрать, какой игровой автомат выбрать

Эта проблема аналогична рекомендательной системе, которая знает, что популярные элементы обычно хорошие, но может оказаться выгодно посмотреть что-то новое. Это можно использовать не только при выборе между алгоритмами, но и для элементов. Более подробно можно почитать в книге «Statistical Methods in Recommender Systems» Дипака К. Агарвала и др. (Cambridge University Press, 2016)

Использование/исследование также используется в Yahoo! News, когда они вводят новый контент, чтобы он не оставался холодным (невидимым). Но тут сначала проводится тестирование среди пользователей первым, позволяя системе понять, какие ответы пользователей являются новым контентом.

9.12.1. Петли обратной связи

В совместной фильтрации для создания рекомендаций используется поведение пользователей, но, когда вы ставите систему в производство, вы должны спросить – как рекомендатор вообще соотносится с поведением пользователя. Если рекомендатор работает, это может помешать разнообразию, как описано в разделе 9.4.2. NIPS (Конференция по нейросетевой обработке информации) 2016 года, где показали интересный документ, демонстрирующий, как измерить такое разнообразие с помощью обратной связи[1].

Держать петли обратной связи в голове весьма полезно, поскольку это позволяет обеспечить, что у пользователей будут альтернативные способы предоставления данных для тестов. На рис. 9.18 цикл показывает, что происходит, если пользователи потребляют только элементы, которые выводятся в рекомендациях. В Netflix должны это учитывать, потому что все рекомендации находятся на их платформе[2]. Вам необходимо каким-то образом ввести новые элементы в

[1] Синха, Аян и др. «Развертка петли обратной связи в рекомендательных системах», **papers.nips.cc/paper/6283-deconvolving-feedback-loops-in-recommender-systems.pdf**.

[2] Посмотрите на слайды **mng.bz/0gw4**.

цикл.

На рис. 9.18 *X* может означать поисковый запрос, какой-то вручную добавленный контент или вообще случайные элементы, как мы говорили в предыдущем разделе. Помните, что, если вам нужно разнообразие, необходимо разрешить пользователям тоже быть разнообразными.

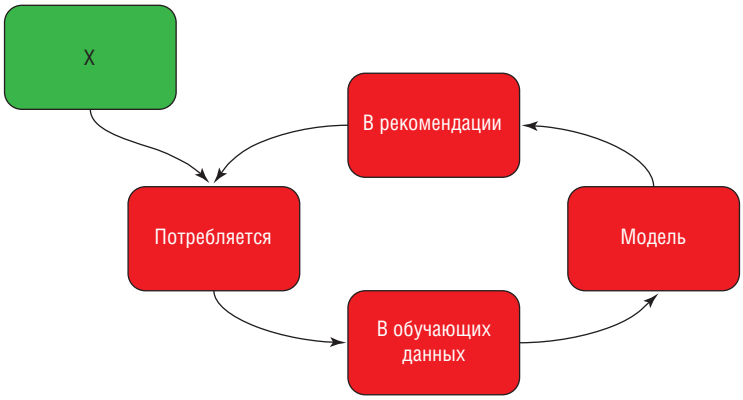


Рис. 9.18. Петли обратной связи рекомендатора

Рекомендательная система – это всегда много вариантов. Вы можете измерять мощность предсказания или ранг. Вы можете разделить ваши данные разными способами: случайным образом, по времени, по пользователям. Удивительно, что мы потратили почти целую книгу на разговоры о том, как реализовать рекомендательную систему, а теперь выясняется, что оценить ее работу тоже трудно? Помните, что трудно запустить систему и оценить, хороша ли она.

Резюме

- Перед внедрением рекомендательной системы нужно проводить тестирование.
- Регрессионный тест позволяет защитить код от ошибок, добавленных случайно.
- Важно, чтобы пользователи находили в рекомендациях элементы, которые бы им неожиданно понравились. Это трудно измерить, но это очень важно.
- Для расчета качества работы рекомендатора или сравнения с предыдущей редакцией используются различные показатели.
- A/B-тестирование (когда посетители делятся на две группы) – это нужная вещь, если вы хотите точно настроить вашу рекомендательную систему. A/B-тестирование позволяет, например, проверить, какие параметры лучше: размер окрестности в совместной фильтрации или количество латентных факторов в матричной факторизации.
- Исследование/использование – это очень важно, если вы хотите оптимизировать вашу систему, когда она запущена.
- Следите за петлями обратной связи.

Глава 10

Фильтрация по контенту

В этой главе пойдет речь о контенте и вкусах пользователей:

- вы познакомитесь с фильтрацией по контенту;
- узнаете, как построить профили пользователей и контента;
- научитесь извлекать информацию из описаний, используя частотность термина/обратную частоту документа (TF-IDF) и скрытое распределение Дирихле (LDA) для создания профилей контента;
- осуществите фильтрацию контента на основе описаний фильмов на сайте Movie-GEEKs.

Из предыдущих глав мы узнали, что можно генерировать рекомендации, имея информацию о взаимодействии между пользователями и контентом (например, анализ списка покупок или совместная фильтрация). Эти методы неплохо работают, но что мы знаем о контенте? У фильма могут быть такие характеристики, как жанр, актеры и режиссеры. В случае с товарами это может быть размер одежды, цвет, данные двигателя автомобиля. Хорошо ли работает рекомендательная система, которая не учитывает все эти вещи?

Ответ – «ДА!», и в предыдущих главах мы уже видели, что все работает, но все-таки кажется, что мы упускаем какую-то информацию. Мы постараемся исправить это, и в данной главе поговорим о том, что вы знаете о контенте и вкусах пользователей.

К концу этой главы вы узнаете, как создать рекомендатор, работающий на основе контента, и даже создадите его. Мы рассмотрим особенности отбора характеристик и обработки текста и эти характеристики будем использовать для фильтрации содержимого. Мы также узнаем два различных алгоритма: *частотность термина/обратная частота документа* (TF-IDF) и *скрытое распределение Дирихле* (LDA). Звучит захватывающе, не так ли? Давайте сразу начнем с примера.

10.1. Описательный пример

Среднестатистический разговор о кино выглядит как-то так:

Я: Я тут посмотрел фильм «Из машины» (хотя я не смотрел, но собираюсь).
Воображаемый собеседник: Ничего себе, и как – понравилось?

Я: Да, некоторые моменты были очень интересны (представим, что я смотрел его).
Воображаемый собеседник: Получается, ты любишь роботов.
Я: Ну да (хотя я не уверен).
Воображаемый собеседник: Хм… если говорить о технологиях, которые выходят из-под контроля, тебе наверняка понравится «Терминатор».
Я: Да (с облегчением).

Что это вообще было? Воображаемый собеседник подумал, к какой категории относится «Из машины», придумал категорию «роботы сходят с ума», мысленно пролистал другие фильмы в этом жанре (хотя такого жанра нет, так что давайте называть это *категорией*) и выбрал «Терминатора». В этой главе мы должны реализовать рекомендатор, который будет делать то же самое.

Вы можете использовать фильтрацию по контенту, чтобы создавать рекомендации из аналогичных элементов, которые часто бывают в разделе **More Like This** (Похожее) (см. рис. 10.1), или генерировать персональные рекомендации, основанные на вкусах пользователя.

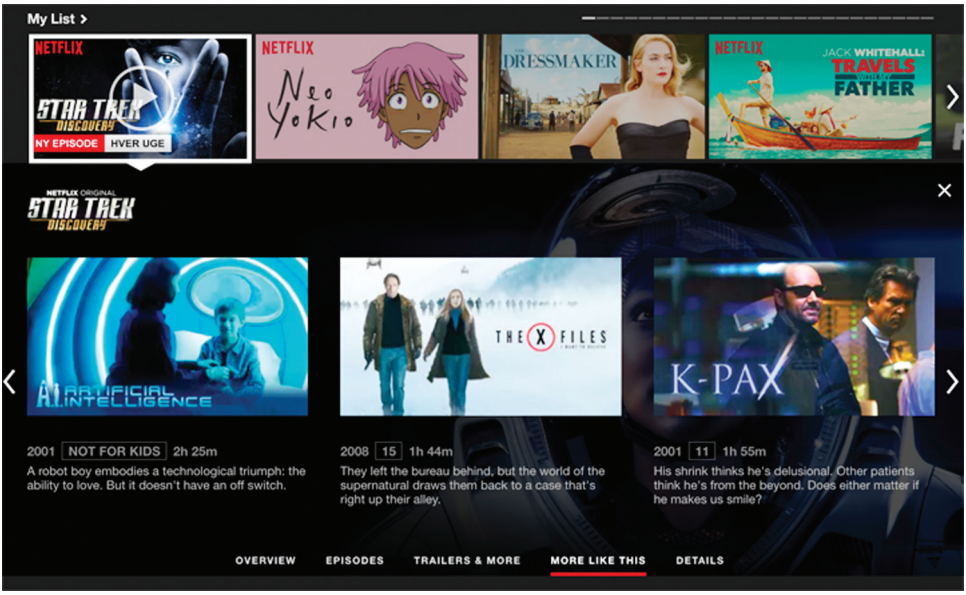


Рис. 10.1. Раздел **More Like This** на сайте Netflix

Воображаемый собеседник, предлагая нам фильм, не опирался на какое-либо мнение о том, хорош ли фильм, а опирался только на метаданные контента и дал рекомендации именно по этому принципу. Сам процесс выглядит примерно как на рис. 10.2.

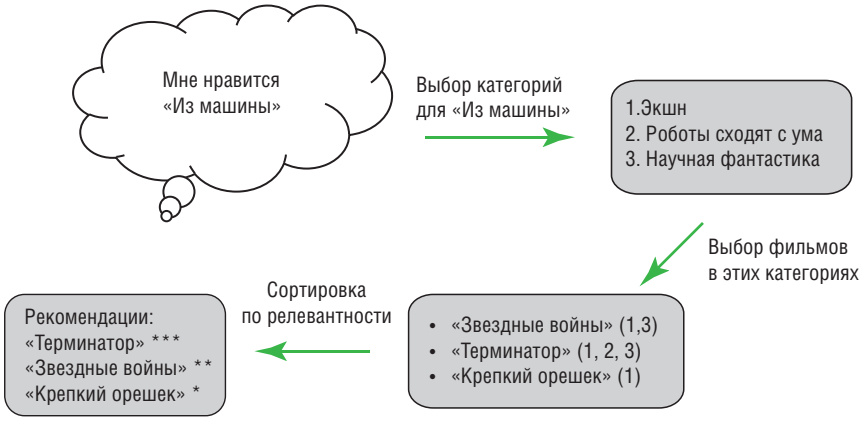


Рис. 10.2. Пример генерации рекомендаций по контенту

Это кажется очевидным (по крайней мере, мне и моему воображаемому другу), но для реализации рекомендательной системы нужно сделать примерно то же самое. На рис. 10.2 мы взяли один фильм («Из машины») и подобрали рекомендаци по нему. Чтобы сделать это, нам нужно как-то выделить контент, который пользователи сочтут похожим. Это может быть немного трудно, потому что люди мыслят своеобразно. Но мы попытаемся.

Сначала поподробнее поговорим о том, что такое фильтрация контента в контексте поиска подходящих слов в алгоритме TF-IDF. С помощью пары трюков мы выделим особенности и построим модель, в которой подобные элементы будут близко. В этой главе мы проделаем долгий путь, так что давайте посмотрим, с чего можно начать, чтобы вам стало понятно хотя бы направление. Начнем с обзора того, что такое фильтрация по контенту. Затем рассмотрим несколько способов описать контент.

Теги – это такая штука, которая часто используется в интернете. Они пришли к нам из социального интернета, или Web 2.0 (и существовали там некоторое время). Теги позволяют пользователям сайта добавлять к вашему контенту ключевые слова. Пример этого можно найти на **imdb.com**, как показано на рис. 10.3. Другой способ описания контента – это текстовые описания, и именно их мы обсудим.

Трудно заставить компьютер понимать сухой текст. Заставить компьютер читать текст – это задача, достойная отдельного научного исследования, и, к сожалению, в эту книгу оно не влезет. Мы будем вести речь об обработке естественного языка (NLP – Natural Language Procesing), и это весьма популярная тема в наши дни.

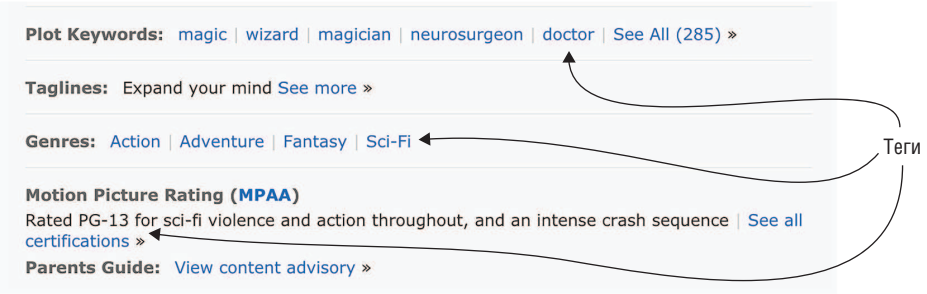


Рис. 10.3. Снимок сайта imdb.com с тегами

Если вам интересно, можно начать с книги «Обработка естественного языка в действии»[1]. Если вы собираетесь рекомендовать текстовый контент, например статьи или книги, стоит изучить NLP. Если вы рекомендуете контент, у которого есть только краткое описание, я не уверен, что это будет стоить приложенных усилий. NLP – это очень большая тема, и большинство из того, что описывается в этой главе, также считается NLP.

Поскольку мы пытаемся научить компьютер извлекать ключевые слова, а не читать текст, нам нужно научиться отделять зерна от плевел. Удалить шум можно многими способами, и несколько из них мы рассмотрим. Затем вычислим наиболее важные слова, используя метод TF-IDF.

Вы можете использовать найденные важные слова, чтобы создать список категорий (или *тем*), затем определить схожие тенденции в описаниях, используя алгоритм LDA[2]. В этой аббревиатуре слово *скрытый (latent)* означает, что темы не соответствуют какой-либо категории, которые вы знаете, *Дирихле* означает способ описания документов с помощью этих тем, а *распределение (allocation)* означает разделение слов на темы. Описание не совсем точное, но если не углубляться в статистику, то подойдет для запоминания. Когда у вас образуется лучшее понимание этих методов, мы рассмотрим, как они реализуются в приложении MovieGEEKs. Теперь перейдем к фильтрации на основе контента.

10.2. Фильтрация на основе контента

Фильтрация на основе контента кажется немного сложнее, чем совместная фильтрация, поскольку здесь требуется извлекать знания из контента. Мы попытаемся извлечь точные определения каждого элемента контента и представить каждый элемент в виде списка значений. На словах все звучит просто, но задача непростая. На рис. 10.4 показана простая процедура обучения рекомендатора на основе контента (автономная часть), а на рис. 10.5 показано, как эта процедура работает, когда пользователь заходит на ваш сайт (онлайн-часть).

---

[1] «Hobson Lane and Hannes Hapke, Natural Language Processing in Action» (Manning, 2018).

[2] Подробнее об алгоритме LDA см. **jmlr.csail.mit.edu/papers/v3/blei03a.html**.

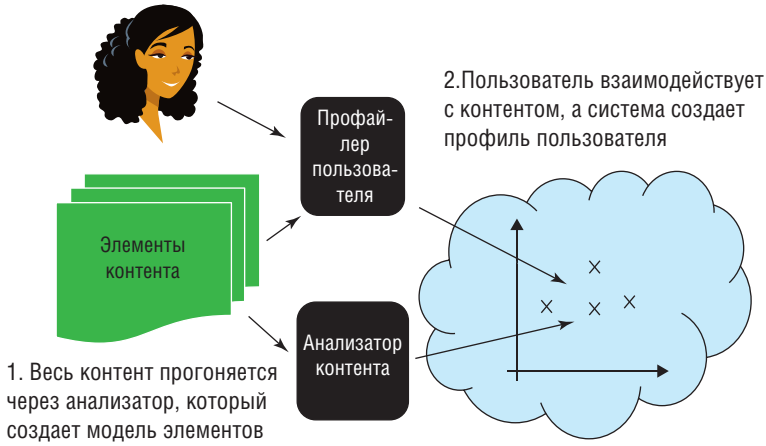


Рис. 10.4. Автономное обучение рекомендатора на основе контента

То есть, чтобы все заработало, необходимо следующее.

Анализатор контента – создает модель на основе контента. В некотором смысле он создает профиль каждого элемента. Здесь выполняется обучение модели.

Профайлер пользователя – создает профиль пользователя. Иногда это просто список элементов, просмотренных пользователем.

Извлекатель элементов – извлекает релевантные элементы, найденные путем сравнения профилей пользователей и профилей элементов, как показано на рис. 10.5. Если профиль пользователя представляет собой список элементов, этот список поэлементно просматривается, и для каждого элемента в списке находятся подобные элементы.

У вас есть несколько способов реализовать эти этапы, и здесь мы поговорим о том, как это работает. Давайте подробнее рассмотрим каждый из трех этапов.

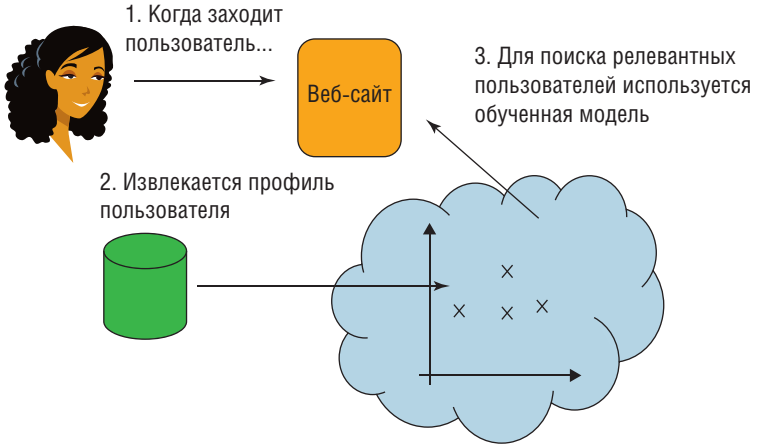


Рис. 10.5. Извлекатель элементов возвращает рекомендации контента во время реальной работы сайта

10.3. Анализатор контента

Анализатор контента – это тот парень, который берет описательные данные о вашем контенте и должен превратить их в какую-то форму, которую сможет использовать компьютер. Для реализации анализатора контента нам нужно определиться с тем, что такое контент и как вы понимаете его. Также мы должны поговорить об извлечении признаков, т. е. тех вещей, которые вы считаете важными для работы вашего алгоритма.

10.3.1. Выделение признаков для профиля элемента

Данные о данных – это *метаданные*. Чтобы избежать путаницы, данные о фильмах мы тоже будем называть метаданными. Метаданные о фильме – это все, что вы можете найти на IMDb: жанр, список актеров и год производства. Сюда же можно отнести, например, тип съемок или стиль одежды, которую носят актеры в фильме. На других сайтах это мог бы быть цвет автомобиля или количество веснушек у женщины на сайте знакомств. Я хотел бы разделить метаданные на два типа:

- факты;
- теги.

Такое деление обычно не используется, но для нас будет удобно представить все именно так. *Факты* – это, например, год производства или состав актеров в фильме, т. е. неоспоримая информация, и вы можете также использовать их в качестве входных данных. *Тегами* могут быть разные вещи, и разные люди будут рассматривать их по-разному.

Социальный интернет сделал популярной фишкой добавление описательных тегов к контенту. Теги могут быть простыми, например «развлекательное», или более узконаправленными, например «разрушение четвертой стены»[1]. Я понятия не имею, что это значит, но в описании «Дедпула» десять человек добавили этот тег, и, видимо, он подходит еще к некоторым фильмам разных жанров и лет[2].

Еще одна проблема с пользовательскими тегами заключается в том, что люди выражают свое мнение по-разному. Для примера посмотрим, что люди говорят о фильме «Джеймс Бонд». Я бы просто поставил тег «Бондиана». Но, глядя на данные, вы увидите, что есть несколько способов описать эти файлы, из которых самый популярный – «007». Чтобы ваша система понимала, что люди на самом деле говорят об одном и том же фильме, в тегах должно использоваться одно и то же слово как можно больше раз. В идеале хотелось бы еще разделить теги, которые несут для людей разный смысл.

> **ПРИМЕЧАНИЕ.** Между фактами и тегами нет четкой границы, но помните, что факты, как правило, неоспоримы и объективны, а теги могут быть немного более субъективными. Вероятно, следует поместить жанры в категорию тегов, но это тоже повод для дискуссии.

[1] Посмотрите на самые популярные фильмы с разрушением четвертой стены в **mng.bz/uC72**.

[2] Подробнее: **mng.bz/997l**.

Самая серьезная проблема разработчиков, пытающихся создать рекомендатор на основе контента, заключается в том, что взять метаданные неоткуда. Какие у вас есть варианты? Вы можете попытаться создать их самостоятельно или поручить другим людям просмотреть контент и пометить его тегами. Но будьте осторожны, так как вы можете получить странные рекомендации. Существуют целые компании, где люди зарабатывают этим на жизнь, – угадайте, где? Давайте рассмотрим пример, в котором вы можете прочувствовать разницу между тегами и фактами.

Теги и факты

«Бэтмен против Супермена: На заре справедливости» (БпС) – это интересный фильм. Скриншот фильма с сайта IMDb показан на рис. 10.6. Глядя на описание фильма, вы можете увидеть, что он относится к жанрам экшн и приключения (и научная фантастика, хотя я не согласен с этим) и что он вышел в 2016 году. Есть и другая информация, например о том, что Бен Аффлек играет Бэтмена. Я бы даже сказал, что это длинный фильм.

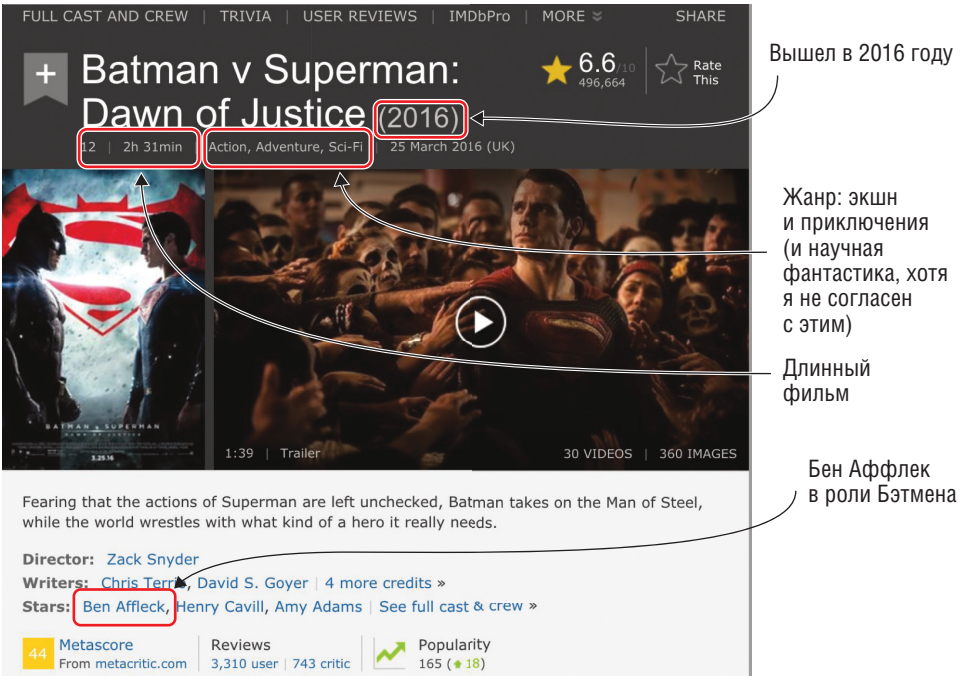


Рис. 10.6. Фильм «Бэтмен против Супермена» на imdb.com

Чтобы представить теги для этого фильма, вы можете составить простой вектор, как показано в табл. 10.1. Вообще, это таблица, но рассмотрим ее как список ключевых значений. В таблице приведены два типа значений:

- двоичные – Бен Аффлек в главной роли, жанр экшн;
- количественные значения – взрывы (если количество взрывов в фильме считать за важный признак) и год выхода.

Таблица 10.1. Профиль фильма «Бэтмен против Супермена»

	Год	Играет Бен Аффлек	Экшн	Приключение	Комедия	Взрывы	Длинный фильм	Собаки	Супергерои
БпС	2016	1	1	1	0	5	1	0	1

В этой таблице почти нет других признаков, кроме тех, что присущи конкретно этому фильму. А вам ведь захочется представить множество различных фильмов, используя один и тот же вектор. И даже комедии. Тут могут быть самые популярные актеры, а также, например, менее значительные вещи: прическа героинь, ванильные сцены, насилие над рыбой и все такое. В конце концов у вас будет длинный список различных признаков и фильмов. При создании рекомендаций на основе контента сложнее всего выяснить, что важно, а что нет.

Теперь, если пользователь купил или лайкнул БпС или другой фильм с Беном Аффлеком, можно сделать вывод о том, что пользователю нравится Бен Аффлек. Затем, зная это, вы можете найти список векторов для других фильмов, в которых играет Бен Аффлек, и рекомендовать их. Но причина лайка может быть и другой. Может быть, пользователь любит необычные жанры фильмов, которые редко используются для классификации фильмов. Этот самый скрытый жанр – то, что мы будем искать, вводя LDA.

10.3.2. Редко встречающиеся данные

Ранее мы говорили об актерах, но мы могли бы подойти к фильму еще более обобщенно и сказать, что в качестве признака может выступать кто угодно, участвовавший в создании фильма. Если мы создаем поиск – это подойдет, но в случае с рекомендатором придется экономить на количестве признаков.

Например, если есть актер, который играет только в одном фильме, здорово, если кому-то фильм понравится, но это не поможет найти похожие фильмы, потому что этот актер больше нигде не играл. Вы не можете использовать упомянутого лишь раз актера для поиска сходства. В этом случае его следует исключить из списка.

10.3.3. Преобразование года в сопоставимую функцию

Как правило, актер появляется только в одной роли в фильме, так что признак «в ролях: *X*» равняется либо 0, либо 1. Если, конечно, мы не говорим об Эдди Мерфи, который обычно играет всех в своих фильмах. Но нужно ли расширять категорию «в ролях: Эдди Мерфи», если он играет пять ролей вместо одной? Если вам нравится Эдди Мерфи, количество его ролей в фильме будет неважно.

Некоторые признаки должны иметь порядковые номера, позволяющие определить, какой элемент ищут чаще, а какой – реже. Например, год выхода. Если пользователь любит фильмы 1980 года, то фильм 1981 года, вероятно, будет ближе к его вкусу, чем фильм 2000 года, поэтому для года выпуска лучше оставить порядковый номер.

При добавлении порядковой функции необходимо определить, насколько эта функция важна. Если поместить порядковую функцию, например год производства, в систему, где все значения лежат между 0 и 1, 2000 – это многовато, даже если это год выпуска. Важно нормализовать или масштабировать данные так, чтобы они лежали между нулем и единицей или близко к этому.

Один из способов сравнить фильмы – построить их на графике, как показано на рис. 10.7. Легко видеть, что, хоть Харрисон Форд не снимался в «Гарри Поттере», а «Звездные войны» (1979) и «Гарри Поттер» (2001) вышли с разницей в 22 года, эти фильмы похожи на рис. 10.7 (сверху), а на рис. 10.7 (внизу) год был нормализован[1]. Можно даже вычесть 1894 (год старейшего фильма в базе данных) из всех годов выпуска или из самого раннего года выпуска, чтобы разница стала еще больше. Когда все данные имеют значения между 0 и 1, заметить разницу будет легко.

Давайте двигаться дальше. Кроме тегов и фактов, в качестве входных данных для вашего алгоритма вы можете использовать описания.

10.4. Извлечение метаданных из описаний

В контексте фильтрации на основе контента новостные статьи интересны тем, что они часто бывают актуальны лишь в течение короткого промежутка времени, а это означает, что их трудно рекомендовать с использованием совместной фильтрации (ну вы помните такую штуку, была в главе 8). Но мы все равно хотим составить рекомендации.

Кроме того, используя параметр популярности, вы можете проанализировать контент. Один из способов сделать это – просмотреть слова в статье, подсчитать, сколько раз встречается каждое из них и как часто эти слова появляются во всех новостях в базе данных. Это можно сделать с помощью TF-IDF, о которой мы поговорим в ближайшее время. Статья представляет собой текст, поэтому контент находится в самой статье, а у фильмов есть написанные кем-то описания.

10.4.1. Составление описаний

Придумать хорошее описание контента нелегко, и здесь важно качество. Расставление тегов для фильмов труднее, чем, например, в случае с книгами или телепередачами, но еще труднее заставить компьютер читать и понимать текст. Прежде чем извлечь информацию из описаний, вы должны убрать все лишнее, что может запутать машину. В следующих разделах мы поговорим именно об этом. Мы поделим текст и будем использовать его по частям.

[1] При делении текущего года на наибольший получаем: 2001 => 1 и 1979 => 1979/2001 = 0,99.

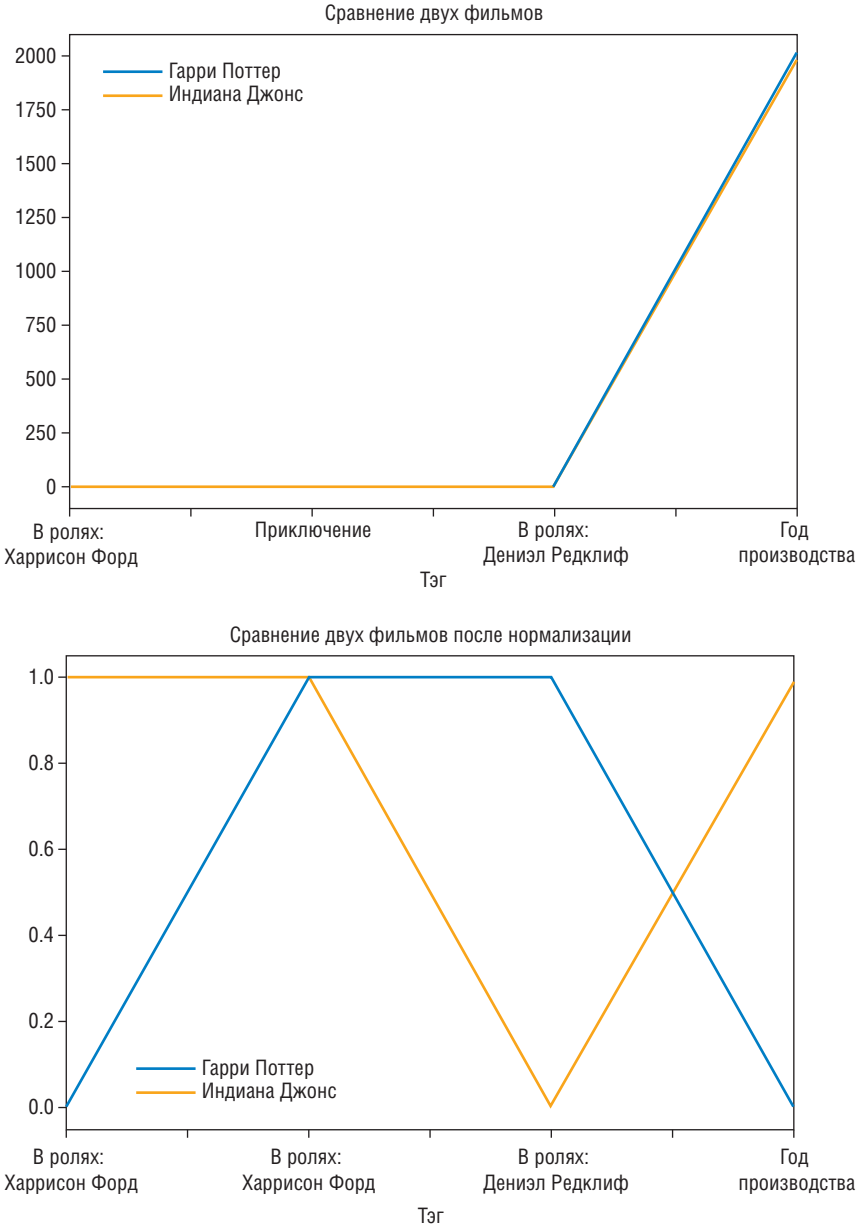


Рис. 10.7. На графиках видно, почему данные нужно нормализовывать. Если год производства не нормирован, все линии будут сливаться в нижней части. Но если год производства делить на наивысший год, все становится хорошо видно

Словарный запас (СЗ) и лексемизация

Чтобы использовать описание, в первую очередь необходимо составить модель «словаря», или словарный запас. Это означает, что вы разделяете описание на массив слов:

«мужчина ест большое крем-брюле» –> [«мужчина», «ест», «большое», «крем», «брюле»]

Обратите внимание, что при использовании СЗ вы теряете часть информации, так как слова «крем» и «брюле» имеют нормальное значение только вместе, а отдельно – совсем другое. Массив получится точно таким же для любой перестановки используемых слов, например «большой крем брюле ест мужчин», даже если смысл всей фразы меняется.

Некоторые слова не добавляют в СЗ знаний – эти слова называются *игнорируемыми словами*. Рассмотрим, как удалить их.

Удаление игнорируемых слов

В описаниях много слов-заполнителей, которые нужны только для того, чтобы сделать удобочитаемое описание. Но если вы разберете документ на массив слов, вы потеряете значение, например, предлогов «в» или «за», поскольку они не несут описательной информации сами по себе. Слова, которые в вашей модели не нужны – это *игнорируемые* слова.

Следующим шагом является удаление игнорируемых слов из СЗ. Список стоп-слов зависит от языка и конкретной задачи. Мы обсудим общие методики, которые подойдут и для русского, и для многих других языков (например, английского). Выполнив в консоли команду, показанную в листинге 10.1, загрузите и установите пакет команд.

ЛИСТИНГ 10.1. Установка пакета игнорируемых слов

```
>Pip3 install stop-words
```

После установки пакета вы можете получить список игнорируемых слов, импортировав `get_stop_words`, как показано в следующем листинге.

ЛИСТИНГ 10.2. Импорт пакета игнорируемых слов

```
from stop_words import get_stop_words
```

Это даст вам массив слов, который, вероятно, не будет нужен в вашей модели. Вызвав `get_stop_words('en')`, вы получите список слов, показанных в листинге.

ЛИСТИНГ 10.3. Английские игнорируемые слова

```
['a', 'about', 'above', 'after', 'again', 'against', 'all', 'am', 'an', 'and', 'any',
'and', 'any', 'are', "aren't", 'as', 'at', 'be', 'because', 'been', 'before',
'being', 'below', 'between', 'both', 'but', 'by', "can't", 'cannot', 'could',
"couldn't", 'did', "didn't", 'do', 'does', "doesn't", 'doing', "don't", 'down',
'during', 'each', 'few', 'for', 'from', 'further', 'had', "hadn't", 'has', "hasn't",
'have', "haven't", 'having', 'he', "he'd", "he'll", "he's", 'her', 'here',
```

```
"here's", 'hers', 'herself', 'him', 'himself', 'his', 'how', "how's", 'i', "i'd",
"i'll", "i'm", "i've", 'if', 'in', 'into', 'is', "isn't", 'it', "it's", 'its',
'itself', "let's", 'me', 'more', 'most', "mustn't", 'my', 'myself', 'no', 'nor',
'not', 'of', 'off', 'on', 'once', 'only', 'or', 'other', 'ought', 'our', 'ours',
'ourselves', 'out', 'over', 'own', 'same', "shan't", 'she', "she'd", "she'll",
"she's", 'should', "shouldn't", 'so', 'some', 'such', 'than', 'that', "that's",
'the', 'their', 'theirs', 'them', 'themselves', 'then', 'there', "there's", 'these',
'they', "they'd", "they'll", "they're", "they've", 'this', 'those', 'through',
'to', 'too', 'under', 'until', 'up', 'very', 'was', "wasn't", 'we', "we'd", "we'll",
"we're", "we've", 'were', "weren't", 'what', "what's", 'when', "when's", 'where',
"where's", 'which', 'while', 'who', "who's", 'whom', 'why', "why's", 'with', "won't",
'would', "wouldn't", 'you', "you'd", "you'll", "you're", "you've", 'your', 'yours',
'yourself', 'yourselves']
```

Вы, вероятно, захотите добавить еще какие-нибудь слова, но для начала подойдет и этот список. Перед использованием модели СЗ вам нужно проверить каждое слово вашего описания на предмет того, игнорируемое ли это слово, и затем удалить/оставить его. Наверняка вы также захотите убрать из вашей модели уничижительные слова (сексизмы или оскорбления).

Убираем минимумы и максимумы

Также стоит обратить внимание на слова, которые появляются во всех описаниях, а также на те, которые появляются несколько или всего один раз в каждом документе. Высокочастотные слова создают фоновый шум, а низкочастотные – усложняют модель, не добавляя при этом пользы. Самые низкочастотные слова также можно убрать, но, удалив слишком много слов, вы рискуете, что модель не сможет выявить нюансы в текстах. Правильное количество может быть разным в разных наборах данных.

Парадигма и лемматизация

В модели СЗ «кот» и «котик» – это разные слова, но устраивает ли нас это? Есть несколько способов «нормализовать» это слово. Самый удобный – использовать лемматизатор, который будет определять корень слова.

У слов «кот» и «котик» общий корень *кот*. Таким образом, слова «кот», «котик» или «котище» будет легко обработать, так как они все начинаются с общего корня. Если вам этого достаточно, можете использовать парадигматический модуль вместо лемматизатора.

Парадигматический поиск – это эвристический процесс, который отрезает концы слов, чтобы достичь нужной цели, и как правило, работает хорошо. Лемматизация означает «сделать все правильно»[1]. Эффективность обоих методов зависит от того, с каким текстом вы работаете. Парадигматический модуль хорош в случаях, где есть слова, которые при разделении остаются теми же, хотя не должны были. Обычно считается, что преимущества перевеши-

[1] В терминах Кристофера Мэннинга и соавт. из «Введения в информационный поиск». Доступно в интернете по адресу **mng.bz/OXR6**.

вают недостатки. Я не советую пользоваться этим на коротких документах, так как этот метод удаляет некоторую информацию. Самое лучшее, что нужно сделать, – это сравнить работу лемматизатора и парадигматического модуля.

Перед тем как двигаться дальше, давайте определимся с направлением движения. Сейчас вам уже должно быть ясно, что такое фильтрация контента и как выполняется выделение признаков. Мы попробуем первый из двух способов выделения признаков из описания и создадим что-то, что можно сравнить с помощью компьютера. Мы попробуем TF-IDF, а затем LDA. Это лишь два из множества способов выделения признаков из текста.

10.5. Поиск важных слов методом TF-IDF

Изучая документы, которые предстоит фильтровать или прогонять поиском, вы смотрите, какие слова или фразы есть в документах. Но помимо игнорируемых слов, в документах много перегруженных слов, которые не добавляют тексту ничего описательного.

Предположим, что вы разрезали эту книгу на мелкие кусочки. У вас получится 100 500 кусочков со словом «рекомендатор», поэтому, даже если это мегаважное слово, это особо не поможет делу. Если у вас есть коллекция статей о достижениях в компьютерной технике, там может быть всего одна статья о рекомендаторах, и слово «рекомендатор» будет определяющим для этой статьи, особенно если оно встречается много раз. Это называется частотностью термина (TF в уравнении), и простой способ определить ее выглядит следующим образом:

tf(слово, документ) = сколько раз слово встречается в документе.

Но чаще используется эта формула:

tf(слово, документ) =1 + log(частота слова).

Чем больше раз слово появляется в одном документе, тем выше вероятность того, что оно важное (при условии, что вы удалили все игнорируемые слова). Но, как уже упоминалось ранее, это важно только в случае, если слово присутствует лишь в нескольких документах. Чтобы определить это, вы можете использовать обратную частоту документа (IDF в уравнении) – число всех документов, поделенное на количество документов, которые содержат данное слово. Вместе мы получаем TF-IDF:

tf-idf(слово, документ) = tf(слово, документ)×idf(слово, документ).

Посмотрите на следующие слова (давайте представим, что каждая строка текста – это отдельный документ):

- у супергероя Дэдпула есть мощная регенерация;
- супергерой Бэтмен сражается с супергероем Суперменом;
- супергерой LEGO-Бэтмен выходит на сцену;
- приключения LEGO-супергероев;
- LEGO-Халк.

Если вы хотите определить важность слова «супергерой», можете использовать расчеты, показанные в табл. 10.2.

Таблица 10.2. TF-IDF слова «супергерой». Парадигматический поиск мы не использовали, так что слово «супергероев» в четвертой строке не учитывается

		TF слова «супергерой»	IDF слова «супергерой»	TF×IDF
1	Супергерой Дэдпул обладает мощной регенерацией	1	5/3	1,66
2	Супергерой Бэтмен и его противник – супергерой Супермен	2	5/3	3,33
3	Супергерой LEGO-Бэтмен выходит на сцену	1	5/3	1,66
4	Приключения LEGO-супергероев	0	5/3	0
5	LEGO-Халк	0	5/3	0

Если кто-то ищет на вашем сайте фильм о супергероях, вы можете показать ему документ 2, у которого значение TF-IDF (Супергерой) равняется 3,33. Затем можно показать два других документа с ненулевыми значениями. Попробуйте самостоятельно рассчитать TF-IDF (Халк)[1].

К чему все эти разговоры о словах, если нам нужны признаки контента? Дело в том, что, когда мы находим слова, которые имеют большое значение TF-IDF, эти слова можно добавить в список признаков. Если рассмотренные нами предложения были бы реальными описаниями фильмов, то слово «супергерой» можно было бы добавить в список признаков вместе со значением TF-IDF.

Это были простые формулы. Но есть поправочка. Обычно для IDF используется формула:

$$idf\left(термин\right) = \log \frac{Всего\ документов}{Количество\ документов\ с\ термином}.$$

ПРИМЕЧАНИЕ. Такая формула дает более устойчивый результат, так как значения логарифма чисел от 1 до 10 близки друг к другу.

Модель TF-IDF была лучшей в своем роде, но после изобретения модели LDA она выпала из милости. В настоящее время все используют LDA или аналогичные модели. Однако, прежде чем вы будете кидать в меня тапками за бесполезную информацию, отметим, что TF-IDF позволяет получить хорошие входные данные для LDA. TF-IDF является классическим и широко используемым методом, но и новые алгоритмы также набирают обороты. Один из них называется Okapi BM25[2].

---

[1] TF-IDF слова «Халк» 0 для всех, кроме строки 5, где 1(5 / 1) = 5.

[2] Дополнительно: см. **en.wikipedia.org/wiki/Okapi_BM25**.

10.6. Моделирование темы с использованием LDA

Если вы спец по машинному обучению, вы, наверное, слышали о фантастических моделях LDA (рис. 10.8), которые позволяют решить все вопросы, касающиеся текста. Хорошо, word2vec тоже популярная штука, так что давайте скажем, что LDA является одним из популярных методов[1]. После прочтения описания word2vec и LDA возникает ощущение, что возможности этих моделей бесконечны! Word2vec подробно описывается в главе 6 книги «Natural Language Processing in Action» (Manning, 2018) Хобсона Лейна и др.

Если вы не читали книгу, то сосредоточьтесь, потому что будет немного сложно. К счастью, трудностей с LDA можно избежать, потому что существует множество библиотек, которые позволяют легко и быстро писать и запускать код. И через мгновение возникнут проблемы, потому что, даже если вы и достигнете нужного функционала, разобраться в нем будет сложно.

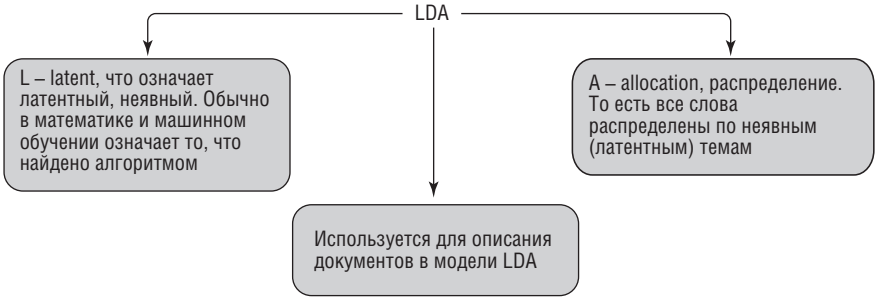


Рис. 10.8. Модель LDA (сокращение от «латентного распределения Дирихле»)

Давайте окунемся в теорию о том, как работает LDA, а затем вернемся к принципам работы с этой моделью. LDA – это так называемая *генеративная модель*. Рассмотрим пример, объясняющий, что это такое.

Пример генеративной модели

Этот детский пример, надеюсь, поможет вам понять генеративные модели. У вас есть три сумки, в каждой из них лежат фигурки одного цвета: красного, синего, зеленого. Чтобы выстроить их в ряд, как показано на рис. 10.9, вам нужно трижды залезть в красный мешок (треугольники), два раза в синий (квадраты), и, наконец, пять раз в зеленый (кружки).

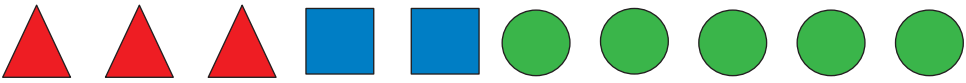


Рис. 10.9. Ряд фигур в примере генеративной модели

Есть и другой способ сгенерировать такой ряд: надо сказать, что из красного мешка мы достаем фигурки 30 % времени, из синего мешка – 20 % времени,

[1] Помните, что модели LDA и word2vec обычно используются для разных целей.

и из зеленого мешка – 50 % времени. То есть ряд *X* на рис. 10.9 формируется по следующему уравнению:

$$x = 0{,}3 \times красный + 0{,}2 \times синий + 0{,}5 \times зеленый.$$

Если бы это был рецепт для создания документов, у вас бы получился документ, показанный на рис. 10.10.

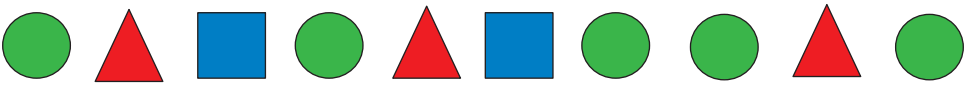


Рис. 10.10. Другой ряд, полученный по рецепту x = **0,3 × красный + 0,2 × синий + 0,5 × зеленый**

Теперь посмотрим на ряд на рис. 10.11.

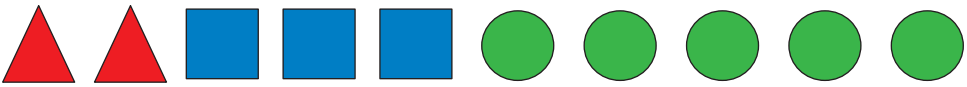


Рис. 10.11. Еще один ряд, полученный по рецепту x = **0,2 × красный + 0,3 × синий + 0,5 × зеленый**

Единственное различие между первым и третьим рядом заключается в одной замене красного треугольника на синий квадрат. Если представить, что каждая из этих фигур – это часть ваших описаний, то идея заключается в том, что вы можете написать формулу, которая облегчила бы сравнение документов. Запомните этот мысленный образ мешков с фигурками, используемых для генерации рядов.

Теперь представим, что вместо мешков у нас будут темы, фигурки – это слова, а строки – описания. Только цвета мы заменим, например, на супергероев, компьютерные науки и еду. Под тему «супергерой» подходят такие слова, как *Человек-паук, летать, сильный, супергерой* и т. д., как показано далее:

- супергерой – Человек-паук, летать, сильный, супергерой;
- компьютерная техника – компьютер, ноутбук, процессор;
- еда – есть, завтрак, вилка.

Если у вас есть описание, показанное ниже:

$$Z = Человек\text{-}паук\ сидит\ дома\ с\ ноутбуком\ и\ ест\ свой\ завтрак\ вилкой,$$

и вы хотите создать его из ваших трех тем, то вы можете взять слово *Человек-паук* из темы о супергероях, *ноутбук* из компьютерной техники, а *есть*, *завтрак*, и *вилку* из темы еды. Это дает вот такую формулу:

$$z = 0{,}2 \times Супергерой + 0{,}1 \times КомпьютернаяТехника + 0{,}3 \times Еда.$$

Все просто, не так ли? Но дело становится чуть сложнее, потому что у каждого слова в теме есть частота, которая говорит о том, насколько это слово важно. Но пока просто отметим это про себя и перейдем к тому, как создавать эти темы.

Пример, связанный с сайтом MovieGEEKs, показан на рис. 10.12. В примере показано, как можно распределить темы, связанные с жанрами кино.

Для примера я взял темы с рис. 10.12, но обычно вся идея моделей тем заключается в том, чтобы компьютер перебирал темы из базы данных описаний.

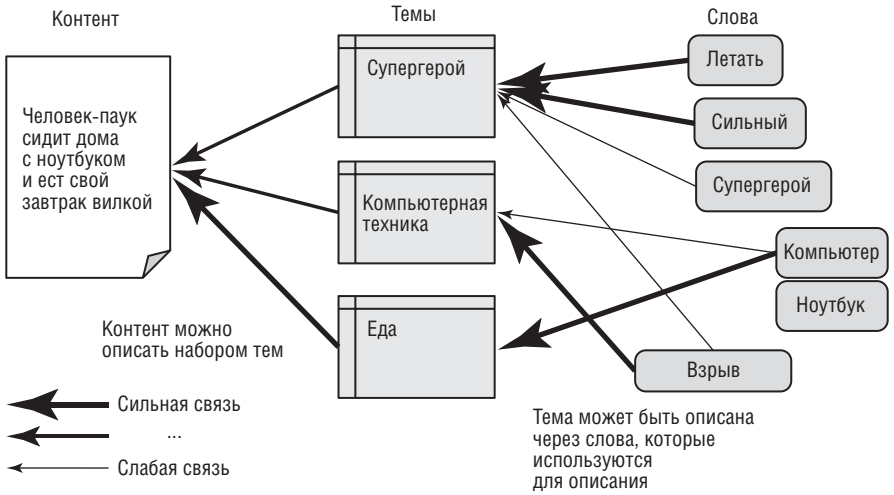


Рис 10.12. Модель тем. Каждая тема определяется списком слов и их соответствующей вероятностью упоминания. Документ может быть описан путем использования формулы с процентами времени на каждую из тем

Генерация тем

Как генерировать темы? На рис. 10.13 показан один из способов сделать это. Идея заключается в том, что вход для алгоритма LDA содержит несколько документов и число *K* – количество тем, которое алгоритм должен создать, чтобы произвести список тем. Выходом алгоритма является список *K* тем и список векторов, по одному документ, в которых содержится вероятность появления каждой темы в документе

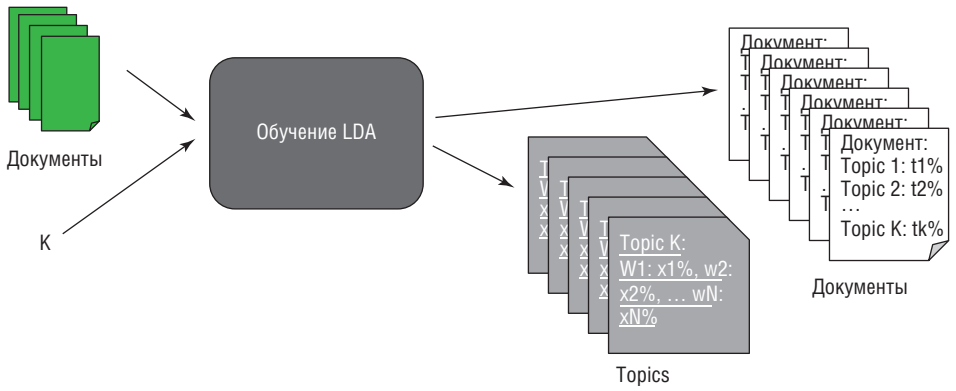


Рис. 10.13. Модель создания темы при работе алгоритма LDA, где на входе имеется список документов и переменная K – число тем для алгоритма. Входные документы могут быть описаны с использованием сформированного списка тем

Темы могут быть получены несколькими различными способами, мы показали лишь один из них. В следующем разделе мы рассмотрим, как соединить слова, темы и документы с помощью выборки Гиббса.

Генерация выборки по схеме Гиббса

Давайте вернемся к рис. 10.13 и подробно остановиться на том, какие у вас есть структуры данных: у вас есть *K* (это количество тем, которые должна содержать модель LDA) и каждый документ, который считается отдельным СЗ (т. е. у вас есть только сами слова, но нет их порядка расположения). Теперь нужно соединить слова с темами и документы с темами. Давайте начнем с тем и слов, показанных на рис. 10.14.

Выстроить список *K* тем и все слова, которые присутствуют в документах, довольно трудно. Но поскольку слова, относящиеся к одной и той же теме, часто встречаются в тех же документах, с этого можно начать. Если вы знаете одно слово, которое присутствует в теме, то у вас есть информация о том, есть ли там еще одно слово. Как раз это и используется в схеме Гиббса[1].

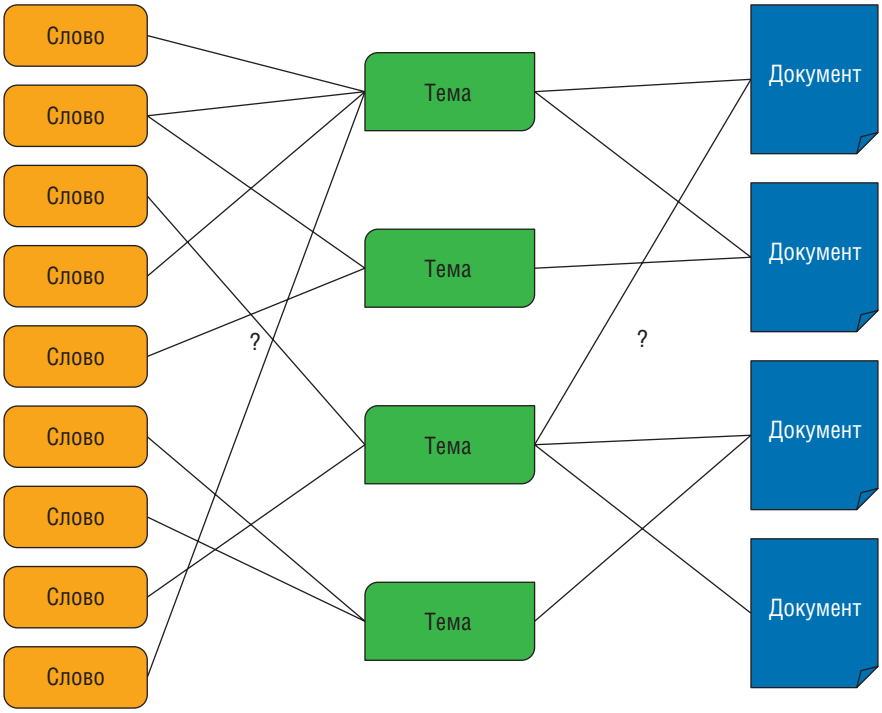


Рис. 10.14. Соединение слов и документов с темами

Схема выборки Гиббса начинается с случайного добавления тем в документы и слов в темы. Представьте, что стрелки на рис. 10.14 расставлены случай-

[1] Филипп Резник и Эрик Хардисти, «Схема выборки Гиббса для непосвященных», www.umiacs.umd.edu/~ htsnik/pubs/LAMP-TR-153.pdf.

ным образом. Схема Гиббса проходит через каждый документ и каждое слово и, учитывая вероятности всех остальных слов в каждом документе, регулирует вероятность этого слова. Это строится на предположении, что темы содержат аналогичные слова. Сопоставляя каждое слово и распределение по-одному, алгоритм Гиббса медленно приближается к распределению слов и тем, которое можно будет адекватно использовать.

Не уверен, что вам нужно читать всю статью, упомянутую в сноске, но я считаю, что вы должны знать, как обучать модель и использовать выборку Гиббса для обучения модели LDA. Угадайте, что будет дальше.

Модель LDA

Выборка Гиббса производит *K* тем, как показано в листинге 10.4. Каждая тема выводится с 10 (K) наиболее вероятными словами в каждой теме. Если в выборке появляются странные слова, такие как 18genre, то это моя попытка ввести жанры в описание. Вместо названий типа *экшн* или *драма* я пронумеровал жанры, чтобы они не путались со словами в описании. Это может повлиять на важность некоторых из этих слов из выборки. В списке (0 …) является первой темой, (1 …) является второй темой и т. д.

ЛИСТИНГ 10.4. Распределение тем, созданное по схеме выборки Гиббса

```
[(0,
  '0.037*18genre + 0.022*love + 0.021*s + 0.012*young + 0.012*man + 0.011*story +
   0.009*woman + 0.008*father + 0.008*35genre + 0.007*10749genre'),
 (1,
  '0.010*vs + 0.010*even + 0.009*nothing + 0.008*based + 0.007*boyfriend + 0.005*s +
   0.005*music + 0.005*rise + 0.004*writer + 0.004*hero'),
 (2,
  '0.012*one + 0.011*s + 0.011*gets + 0.011*three + 0.009*like + 0.008*99genre +
   0.008*years + 0.007*car + 0.006*different + 0.006*marriage'),
 (3,
  '0.013*s + 0.010*28genre + 0.009*house + 0.009*two + 0.009*878genre + 0.008*years +
   0.008*daughter + 0.007*year + 0.007*world + 0.007*old'),
 (4,
  '0.024*35genre + 0.015*girl + 0.011*10749genre + 0.010*year + 0.009*old + 0.009*go
+
   0.009*falls + 0.008*s + 0.008*get + 0.008*four'),
 (5,
  '0.025*53genre + 0.016*80genre + 0.015*27genre + 0.013*28genre + 0.011*18genre +
   0.008*police + 0.008*s + 0.007*goes + 0.007*couple + 0.007*discovers'),
 (6,
  '0.016*t + 0.016*s + 0.011*school + 0.010*friends + 0.010*new + 0.009*boy +
   0.008*first + 0.007*will + 0.007*35genre + 0.007*show'),
 (7,
  '0.013*99genre + 0.008*s + 0.008*fall + 0.007*movie + 0.007*documentary +
0.007*power +
```

```
   0.005*18genre + 0.005*wants + 0.005*will + 0.005*move'),
 (8,
  '0.046*film + 0.011*friend + 0.010*past + 0.009*18genre + 0.009*directed +
   0.007*upcoming + 0.007*produced + 0.006*ways + 0.006*festival + 0.006*turned'),
 (9,
  '0.016*s + 0.014*will + 0.013*one + 0.010*10402genre + 0.009*life + 0.009*journey +
   0.008*game + 0.007*work + 0.007*time + 0.006*world')]
```

Каждая тема содержит все слова, найденные во всех входных документах, но у многих слов такая низкая вероятность (близкая к нулю), что они нам не интересны. Вероятность – это число перед каждым словом в списке. При попытке сгенерировать документ из этих тем это слово будет извлекаться с меньшей вероятностью. Если у вас есть документ, который был создан исключительно из темы 0, то каждое вхождение слова *love* будет иметь вероятность 2,2 %, что и видно в этом списке.

ЛИСТИНГ 10.5. Вероятность слова love в теме 0

```
[(0,
  '0.037*18genre + 0.022*love + 0.021*s + 0.012*young + 0.012*man + 0.011*story +
0.009*woman + 0.008*father + 0.008*35genre + 0.007*10749genre'),
```

Другой документ может быть сформирован путем отбора слов из тем 2, 3 и 9, к примеру. Документ может быть представлен в модели LDA, как показано в следующем листинге.

ЛИСТИНГ 10.6. Распределение тем документа

```
[(2, 0.02075648883076852),
 (3, 0.1812829334788339),
 (9, 0.78545976997831202)]
```

В листинге видно, что 2,1 % времени слово генерируется из темы 2; 18,1 % – из темы 3 и 78,5 % – из темы 9.

Теперь вы взнаете, как это работает, по крайней мере, в теории. Большая часть работы в создании функции LDA приходится на обработку входного текста. Труднее всего понять модель LDA. Вам придется смотреть на входные и выходные данные длительное время и даже вспоминать их во сне, чтобы картинка сложилась.

Вспомним о Википедии

Пока что мы поговорили только о добавлении описаний и документов в LDA, но о каких документах вообще речь? В предыдущем примере и в реализации на MovieGEEKs, о которой мы скоро поговорим, используются одни и те же документы, на которых мы будем вычислять сходство. Хорошей идеей будет использовать набор данных, который описывает (или, по крайней мере, дает) хорошие примеры для нескольких тем.

К примеру, медиакомпания Issuu (**issuu.com**) использует модель LDA, генерируя рекомендации для пользователей о том, что почитать. Он использует данные из Википедии для создания модели с хорошо определенными и понятными темами. У Википедии также есть документы с классификацией для всего, что гарантирует хорошее распределение тем по модели.

Добавление в документ признаков и тегов

Еще раз повторимся: документ представлен в виде C3, при этом информация о порядке и связи слов отсутствует. Если вы хотите управлять темами документа, вы также можете вставлять нужные слова в описания. Например, если вы хотите включить информацию об актерах или годе производства – добавьте ее в C3. Вы видели пример этого в листинге 10.4, где появлялись слова вроде *25genre*. Чуть позже остановимся на этом подробнее.

10.6.1. Какими крутилками настраивать LDA?

Все делается по волшебству. Но работает ли? Посмотрите на рис. 10.15 и убедитесь, что не все тут идеально.

Рис. 10.15. Рекомендации на основе контента из LDA, которые нужно подправить. Мультфильм Диснея, который похож на «Невинность и преступление», – звучит так себе

Качество модели зависит от качества документов, а хорошее качество документов обусловлено примерами типов/жанров/предметов, которые вы хотите видеть в модели. Например, описания фильмов, которые вы используете в этом разделе, не всегда подходят для этого. Описание мультика «В поисках Немо» ничего не говорит о том, что это мультик и что дело происходит в море. Вы можете добавить жанры, которые будут указывать на мультипликацию, но составлять описания вручную – малоприятное занятие, поэтому лучше придумать способы борьбы с недостатком информации.

Вы можете посмотреть, как рекомендатор сработает с фильмом, у которого точно много похожих элементов в каталоге. Например, возьмем фильм о Человеке-пауке, как показано на рисунке. Фильмов о Человеке-пауке много, так что такие рекомендации подойдут разве что самым ярым его фанатам. Помимо входных данных, нужно также подумать о том, сколько тем использовать.

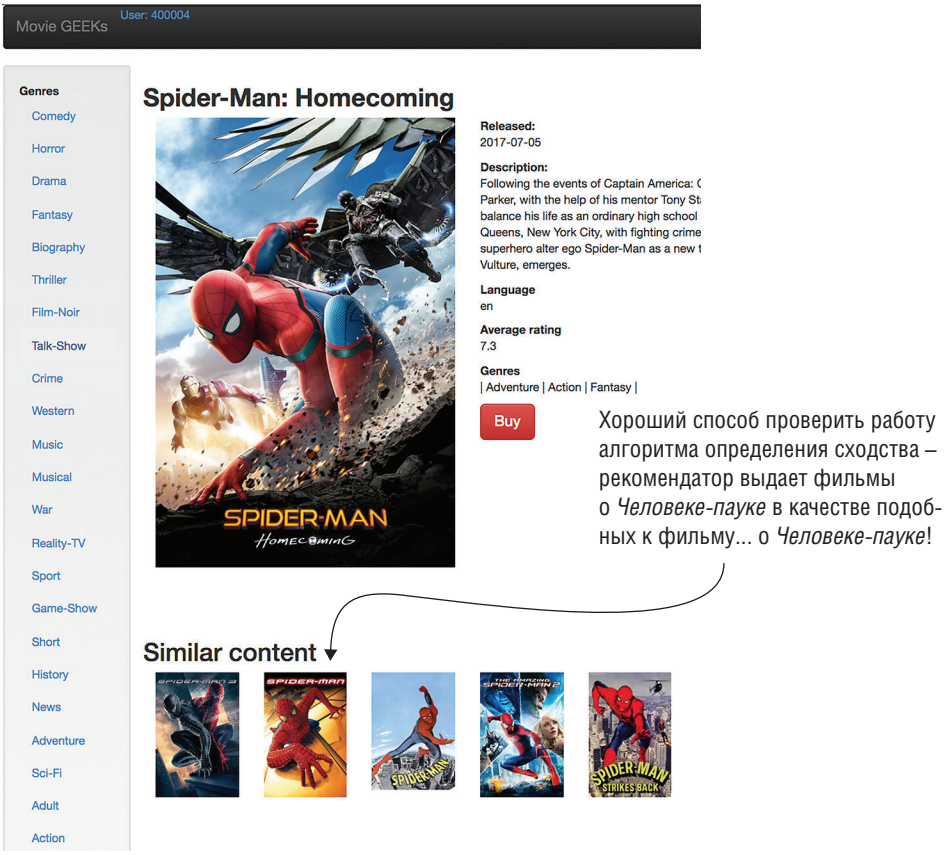


Рис. 10.16. Рекомендатор на основе контента понимает, что все фильмы о Человеке Пауке похожи

Сколько нужно тем?

Чтобы сделать идеальную модель LDA или безвозвратно испортить ее, нужно выбрать определенное количество тем. Это своего рода искусство, потому что нет никакого способа проверить, правильное ли вы выбрали количество.

В аналитической части сайта MovieGEEKs (**mng.bz/04k5**) представлена интерактивная визуализацию модели (**http://127.0.0.1:8001/analytics/lda**), показанная на рис. 10.17. Каждый круг соответствует теме. Наведя курсор на тему, вы получите список наиболее часто используемых слов в этой теме.

Здесь важно выделить список тем, которые распределены равномерно. Если выбрать слишком низкое значение *K*, многие документы будут похожи друг на друга, как если бы вы описали все фильмы, используя только жанры комедии и драмы. Но если вы выбираете слишком много тем, у вас может и не оказаться подобных документов – все они будут казаться разными. И здесь мы возвращаемся к проклятию размерности.

Один из способов лучше визуализировать вашу модель и понять, хорошо ли она распределяет темы, – это использовать инструмент под названием pyLDAvis (рис 10.17).

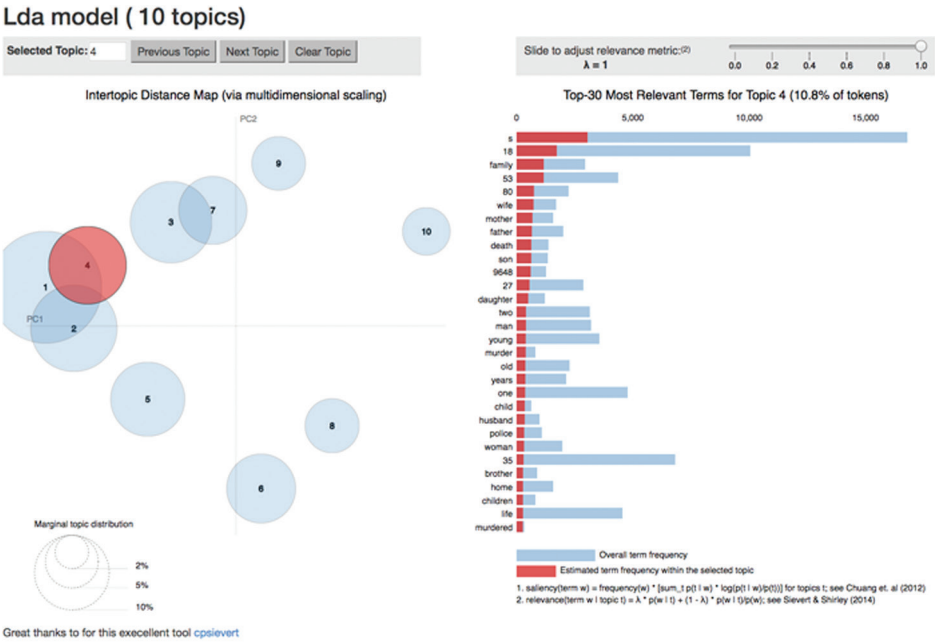


Рис. 10.17. Панель pyLDAvis, где выводится распределение тем

Панель pyLDAvis была создана с использованием библиотеки Python под названием pyLDAvis[1], как описано в работе Карсона Зиверта и Кеннета Е. Ширли[2].

[1] Документацию по pyLDAvis можно посмотреть по ссылке **pyldavis.readthedocs.io/en/latest/**.

[2] Карсон Сиверт и Кеннет Е. Ширли, «LDAvis: Способ визуализации и интерпретации тем», **mng.bz/qPf7**.

При принятии решения о выборе количества тем для обучения модели LDA хорошо бы проверить следующее:

- проверьте панель (например, как на рис. 10.17) и посмотрите, не накладываются ли темы друг на друга;
- проверьте, выдает ли LDA подобные элементы.

Позже в этой главе вы увидите, как LDA реализована на сайте MovieGEEKs, и сможете проверить, дает ли модель хорошие рекомендации. Моя первая попытка выдала рекомендации, показанные на рис. 10.15.

Как должна выглядеть модель LDA? Интуитивно правильно. Выберите несколько фильмов, которые вам нравятся, и поиграйтесь с настройками, пока модель не заработает. Затем попросите друга сделать то же самое и посмотрите, сможете ли вы настроить ее так, чтобы вы оба были довольны.

Играем с альфа и бета

В работе с LDA у вас есть еще два параметра, которые можно покрутить. Вы можете регулировать параметры альфа и бета для настройки распределения обоих документов и слов в темах.

Если ввести высокое значение альфа, каждый документ будет распределен по многим темам. Низкое значение альфа распространяется только на несколько тем. Преимущество высокого альфа заключается в том, что документы кажутся больше похожими, а если у вас есть узкоспециальные документы, то низкое значение альфа будет разделять на несколько тем.

То же самое относится и к бета: высокое значение бета делает темы более похожими, потому что вероятности будут распределены по большему числу слов, которые используются для описания каждой темы. Например, вместо 10 слов в теме с вероятностью выше 1 % у вас может быть 40 слов. Это дает большее перекрытие. Большинство людей, с которыми я общался, с осторожностью подходят к изменению значений альфа и бета, так как это довольно грубая настройка, но, если есть время поиграть, – на здоровье.

10.7. Поиск подобного контента

Теперь, когда у вас модель LDA, появился новый способ находить подобные элементы. Спроецировав два документа на модель LDA, вы можете рассчитать их сходство. Поскольку распределение вероятностей можно рассматривать как векторы, многие в своих расчетах используют коэффициент Отиаи (одна из функций сходства, о которых мы говорили в главе 7).

В принципе, эту модель LDA можно также использовать для сравнения документов, которые при создании модели не использовались. Это одна из наиболее важных функций для рекомендатора на основе контента в решении задач с «холодными» элементами, о которых мы говорили в главе 6. Это можно использовать для составления списка рекомендаций «Похожий контент».

Позже в этой главе мы подробнее поговорим о реализации модели на MovieGEEKs и приведем пример вычисления сходства двух фильмов. А сейчас да-

вайте посмотрим, как составить персональные рекомендации в системе рекомендатора на основе содержания.

10.8. Создание профиля пользователя

Если вы любите Джеймса Бонда, то вам может понравиться также… В такой простой постановке нам не нужен никакой профиль пользователя. Имея модель LDA или просто вектор признаков, о которых мы говорили ранее, вы найдете вектор для фильма о Джеймсе Бонде, а затем – фильмы с похожими векторами. Но если вы предоставляете персональные рекомендации, необходимо создать профиль пользователя, который охватывает все понравившиеся ему фильмы. Давайте посмотрим, как это сделать с помощью LDA, а затем с помощью TF-IDF.

10.8.1. Создание профиля пользователя с помощью модели LDA

При создании персонализированных рекомендаций вам нужно просмотреть весь список элементов, которые нравятся пользователю, и найти другие элементы, которые также могут ему понравиться. В реальной жизни такая функция не является индивидуальной, так как пользователь не обязательно может быть однозначно определен по потребленному им контенту (даже если большинство систем рекомендаторов, вероятно, дадут один и тот же результат). Но все равно тут будет некоторая доля персонализации.

Вы можете перебрать каждый из элементов, которые нравятся пользователю, и для каждого из них найти подобные элементы. Когда вы получите список, отсортируйте его по произведению значений сходства и оценок пользователей из исходного списка потребленных элементов. Разложим все это на простые этапы:

- берем все потребленные активным пользователем элементы;
- для каждого элемента:
  - находим подобные элементы с использованием модели LDA,
  - рассчитываем оценку на основе сходства оценок активных пользователей;
- сортируем элементы по оценке;
- сортируем по релевантности (если есть соответствующие данные).

Есть и альтернативный способ, описанный Джобином Уилсоном и др., – вы можете создать вектор LDA для пользователя, а затем найти подобные элементы[1]. Вам может быть интересно посмотреть, как это будет работать. Оценка выглядит очень хорошо.

[1] Джобин Уильсон и др., «Улучшение рекомендаторов с совместной фильтрацией с использованием моделирования тем», Февраль 2014 г. Аннотация: **arxiv.org/abs/1402.6238**.

10.8.2. Создание профиля пользователя с помощью модели TF-IDF

Если имеются описанные ранее векторы с тегами и фактами, есть еще один метод, которым вы могли бы создать профиль пользователя. Вы можете агрегировать векторы элементов, которые пользователю нравятся, и вычесть элементы, которые пользователю не нравится. Посмотрите на информацию в табл. 10.3.

Таблица 10.3. Векторное представление нескольких фильмов

	В ролях: Бен Аффлек	Экшн	Приключения	Комедия	Взрывы*
«Бэтмен против Супермена»	1	1	1	0	5
«День Валентина»	1	0	0	1	0
«В поисках утраченного ковчега»	0	1	1	1	2
«Ла-Ла Лэнд»	0	0	0	1	0

* Значения выдуманные

Если у вас есть пользователь (мы тут не выполняем совместную фильтрацию, так что достаточно посмотреть на одного), который оценил «В поисках утраченного ковчега» на пять звезд (кто не любит этот фильм?) и «Ла-Ла Лэнд» на три звезды, вы можете создать профиль пользователя, умножая оценку на вектор фильма и сложив векторы, как показано в табл. 10.4.

Таблица 10.4. Умножаем оценки пользователей на векторы фильмов и складываем элементы, чтобы создать профиль пользователя

	В ролях: Бен Аффлек	Экшн	Приключения	Комедия	Взрывыа
«В поисках утраченного ковчега»	0	5	5	5	10
«Ла-Ла Лэнд»	0	0	0	3	0
Профиль пользователя	0	5	5	8	10

* Значения выдуманные

Теперь вы можете использовать этот вектор, чтобы найти для пользователя подобный контент. Нужно будет нормализовать значения так, чтобы они находились в том же масштабе, что и фильмы, и количество взрывов, вероятно, следует уменьшить, если в фильме их вообще нет. В этом случае профиль будет выглядеть, как показано в табл. 10.5.

Таблица 10.5. Умножив оценки пользователя на векторы фильмов и сложив все элементы, получаем профиль пользователя

	В ролях: Бен Аффлек	Экшн	Приключения	Комедия	Взрывы
Профиль пользователя	0	5	5	8	5

Эта процедура хорошо работает с фильмами. Здесь плюс в том, что вы можете просмотреть фильмы с учетом всех предпочтений пользователя (при условии выбора правильных тегов и фактов, конечно), но, хотя мне нравится шоколад и лазанья, это не значит, что я ем их вместе. То же самое можно сказать и о многих других атрибутах, которые можно придумать для любой еды или кино. В результате вы можете использовать это, чтобы увидеть, что пользователь любит комедии больше, чем экшн и приключения.

В аналитике сайта MovieGEEKs я представлял вкус пользователя, просмотрев все оцененные пользователем фильмы и сложив оценки для каждого жанра, как показано в следующем листинге. Вы можете просмотреть код в файле */analytics/views.py*.

ЛИСТИНГ 10.7. Получение предпочтений из оценок

```
for movie in movies:                          # Р ссчит но для к ждого оцененного пользов телем фильм
    id = movie.movie_id

    rating = ratings[id]                      # Получение оценки

    r = rating.rating
    sum_rating += r
    movie_dtos.append(MovieDto(id, movie.title, r))
    for genre in movie.genres.all():          # Проход всех ж нров и сост вление слов ря с н зв ниями ж нров и суммой оценок

        if genre.name in genres_ratings.keys():
            genres_ratings[genre.name] += r - user_avg
            genres_count[genre.name] += 1

# Поиск м к-сим льных зн чений
max_value = max(genres_ratings.values())
max_value = max(max_value, 1)                 # Проверк , что зн чение не меньше единицы
max_count = max(genres_count.values())
max_count = max(max_count, 1)                 # Проверк , что зн чение не меньше единицы

genres = []
for key, value in genres_ratings.items():     # Норм лиз ция зн чений
    genres.append((key, 'rating', value/max_value))
    genres.append((key, 'count', genres_count[key]/max_count))
```

Этот код используется для создания графиков, как показано на рис. 10.18, для пользователя 100 (**http://localhost:8001/analytics/user/100/**).

10.9. Рекомендации на основе контента на сайте MovieGEEKs

Как уже несколько раз упоминалось, мы пройдем процедуру реализации и использования модели LDA на основе контента. Сперва нужно сказать несколько слов о получении данных.

10.9.1. Загрузка данных

Используемый вами набор данных не содержит описаний фильмов, так что данные извлечем с помощью **www.themoviedb.org**. В корневой папке в файлах книги есть скрипт под названием *populate_sample_of_descriptions.py*, который извлекает описания самых последних фильмов. Пример получаемых данных можно увидеть на рис. 10.19[1].

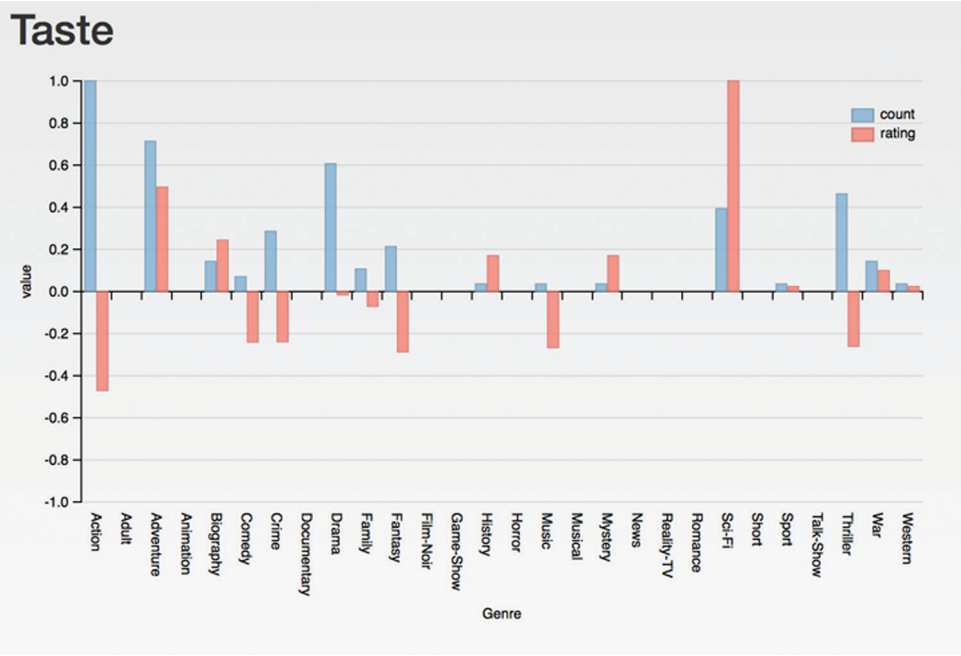


Рис. 10.18. График вкусов для пользователя 100

[1] См. **mng.bz/1UXq**.

```
- {
    poster_path: "/z09QAf8WbZncbitewNk61KYMZsh.jpg",
    adult: false,
    overview: ""Finding Dory" reunites Dory with friends Nemo and Marlin on a search
    for answers about her past. What can she remember? Who are her parents? And where
    did she learn to speak Whale?",
    release_date: "20016-06-16",
  - genre_ids: [
        16,
        10751
    ],
    id: 127380,
    original_title: "Finding Dory",
    original_language: "en",
    title: "Finding Dory",
    backdrop_path: "/iWRKYHTFlsrxQtfQqFOQyceL83P.jpg",
    popularity: 27.117383,
    vote_count: 1234,
    video: false,
    vote_average: 6.69
  },
```


Рис. 10.19. Объект JSON для фильма «В поисках Дори»


```
- {
    poster_path: "/z09QAf8WbZncbitewNk61KYMZsh.jpg",
    adult: false,
    overview: ""Finding Dory" reunites Dory with friends Nemo and Marlin on a search
    for answers about her past. What can she remember? Who are her parents? And where
    did she learn to speak Whale?",
    release_date: "20016-06-16",
  - genre_ids: [
        16,
        10751
    ],
    id: 127380,
    original_title: "Finding Dory",
    original_language: "en",
    title: "Finding Dory",
    backdrop_path: "/iWRKYHTFlsrxQtfQqFOQyceL83P.jpg",
    popularity: 27.117383,
    vote_count: 1234,
    video: false,
    vote_average: 6.69
  },
```


Сценарий в следующем листинге позволяет извлечь и сохранить описания в базу данных. Мы приводим этот сценарий, потому что вы можете поиграть с ним и заставить его работать на текущих настройках. Этот код можно посмотреть в папке */pre/moviegeek/populate_sample_of_de-scriptions.py*.

ЛИСТИНГ 10.8. Извлечение описаний фильмов

```
def get_descriptions(start_date = "1990-01-01"):
    url = """https://api.themoviedb.org/3/discover/movie"""
    qs =
     """?primary_release_date.gte={}&api_key={}&page={}"""
    api_key = get_api_key()

    this_date = start_date
    last_date = ""
    MovieDescriptions.objects.all().delete()
    today_date = str(datetime.now().date())
    errorno = 0

    while today_date > last_date != this_date and errorno < 10:

        for page in tqdm(range(1, NUMBER_OF_PAGES)):
            formated_url = url + qs.format(start_date, api_key, page)
            r = requests.get(formated_url)
            r_json = r.json()
            if 'results' in r_json:
                for film in r_json['results']:
                    id = film['id']
                    md = MovieDescriptions.objects.get_or_create(movie_id=id)[0]

                    md.imdb_id = get_imdb_id(id)
                    if md.imdb_id is not None:
                        md.title = film['title']
                        md.description = film['overview']
                        md.genres = film['genre_ids']
                        last_date = film['release_date']

                        md.save()

            elif 'errors' in r_json:
                print(r_json['errors'])
                      errorno += 1
                break
            time.sleep(1)
```

URL н API фильм . Выводит фильмы с 1990 год по сегодняшний день

Если не сегодня – продолж йте, если ошибок меньше десяти

Прогоняется через 1000 стр ниц (API больше не позволяет)

Проходит через все фильмы н стр нице

Сохр нение в б зу д нных

Если возник ет ошибк , увеличить счетчик ошибок и выйти из цикл

Запрос API фильма содержит данные, которые должны быть больше или равны указанному. В скрипте задано в 1970 году. Чтобы получить описания фильмов, запустите листинг ниже.

ЛИСТИНГ 10.9. Загрузка описаний фильмов

```
$ python populate_sample_of_descriptions.py.
```

Этот код загружает описания, которые можно использовать для модели тем. Могут возникнуть проблемы при запуске, так как описания берутся с **www.themoviedb.org**, а там может быть ограничение на количество запросов. Я добавил в код команду `time.sleep(1)`, которая на одну секунду включает спящий режим между запросами. Этого может оказаться недостаточно, так что, если вы будете получать гневные письма с сервера, откройте файл *populate_sample_of_descriptions.py* и укажите значение побольше.

10.9.2. Обучение модели

Мы будем использовать библиотеку под названием Gensim, в которой есть реализация моделей LDA. Существуют и другие модели, но это одна из самых популярных, и мне понравилось с ней работать.

Вы можете установить Gensim, выполнив команду, указанную в листинге 10.10. Это часть общих требований для сайта MovieGEEKs, так что, если вы установите ее, все будет хорошо (помимо запуска команды `pip3 install -r requirements.txt`)[1].

ЛИСТИНГ 10.10. Загрузка пакета Gensim

```
$ pip3 install gensim.
```

С помощью библиотеки Gensim вам будет несложно создать модель LDA. Вам понадобятся загруженные и «причесанные» документы, как мы уже говорили ранее. Реализация показана в следующем листинге, а код можно посмотреть в файле */Builder/LdaBuilder.py*.

ЛИСТИНГ 10.11. Построение модели LDA Gensim

```
texts = []
tokenizer = RegexpTokenizer(r'\w+')
for d in data:
    raw = d.lower()

    tokens = tokenizer.tokenize(raw)

    stopped_tokens = self.remove_stopwords(tokens)

    stemmed_tokens = stopped_tokens

    texts.append(stemmed_tokens)
```

Проверк к ждого документ

Превр щение з гл вных букв в строчные

Р збиение текст н м ссив

Уд ление игнорируемых слов

П р дигм тическ я коррекция

Доб вление концентриров нного документ

[1] Подробнее об установке Gensim: **radimrehurek.com/gensim/install.html**.

```
dictionary = corpora.Dictionary(texts)
corpus = [dictionary.doc2bow(text) for text in texts]

lda_model = models.ldamodel.LdaModel(corpus=corpus, id2word=dictionary,
                                     num_topics=n_topics)
index = similarities.MatrixSimilarity(corpus)
self.save_lda_model(lda_model, corpus, dictionary)
self.save_similarities(index, docs)
  :
```

Созд ние СЗ

Созд ние совокупности, содерж щей м ссив всех документов, предст вленных в форме СЗ

Отобр жение постро-енной модели LDA

Созд ние м трицы сходств

Сохр нение модели LDA

Сохр нение зн чений сходств в б зу д нных

10.9.3. Создание профилей элементов

Профили элементов создаются с помощью модели и представлены в виде вектора LDA, о котором мы говорили. В показанном ранее коде я решил пойти чуть дальше и создал матрицу сходства напрямую, потому что при подборе рекомендаций нужно просматривать исследуемый элемент. Это не всегда возможно, и у вас, вероятно, будет гораздо больше элементов, чем мы использовали здесь.

В матрице сходства удобно посмотреть на значения сходства. Если все они близки к нулю, то стоит перейти к более общим данным, чтобы соединить больше элементов. Чтобы создать модель и заставить сайт MovieGEEKs выдавать рекомендации на основе контента, запустите файл *LdaBuild.py*, показанный в листинге ниже.

ЛИСТИНГ 10.12. Запуск построения модели LDA

```
$ python -m builder.lda_model_calculator
```

10.9.4. Создание пользовательских профилей

Я определенно повторяюсь: есть много способов создать профиль пользователя. Самый простой, который вы будете использовать в следующем листинге, – это взять список предметов, понравившихся пользователю, и для каждого из них определить подобные элементы. Вы найдете этот фрагмент кода в знакомом сценарии */recs/content_based_recommender.py*.

ЛИСТИНГ 10.13. Генерация рекомендации для пользователя «вживую»

```
def recommend_items(self,
                    user_id,
                    num=6):
    movie_ids = Rating.objects.filter(user_id=user_id)
                              .order_by('-rating')
                              .values_list('movie_id', flat=True)[:100]
```

Используется 100 лучших оценок пользов теля

```
    return self.recommend_items_from_items(movie_ids, num)

def recommend_items_by_ratings(self,
                               user_id,
                               active_user_items,
                               num=6):

    content_sims = dict()

    movie_ids = {movie['movie_id']: movie['rating'] \
                 for movie in active_user_items}
    user_mean = sum(movie_ids.values()) / len(movie_ids)

    sims = LdaSimilarity.objects.filter(Q(source__in=movie_ids.keys())
                                        & ~Q(target__in=movie_ids.keys())
                                        & Q(similarity__gt=self.min_sim))

    sims = sims.order_by('-similarity')[:self.max_candidates]
    recs = dict()
    targets = set(s.target for s in sims)
    for target in targets:

        pre = 0
        sim_sum = 0

        rated_items = [i for i in sims if i.target == target]

        if len(rated_items) > 0:
            for sim_item in rated_items:
                r = Decimal(movie_ids[sim_item.source] - user_mean)
                pre += sim_item.similarity * r
                sim_sum += sim_item.similarity
            if sim_sum > 0:
                recs[target] = \
                    { 'prediction': Decimal(user_mean) + pre/sim_sum,
                      'sim_items': [r.source for r in rated_items]}
    return sorted(recs.items(),
                  key=lambda item: -float(item[1]['prediction']))[:num]
```

Вызов recommend_items_by_rating

Этот метод позволяет облегчить тесты, т к к он приним ет список оценок

Извлечение всех идентифик торов оцененных фильмов

Р счет среднего для пользов теля

В м нужны только элементы для сходств

Сортировк по зн чению сходств и извлечение лучших. Количество зн чений выбир ется из соображений б л нс производительности и скорости

Поиск всех элементов, подобных целевому элементу

Проход через целевые зн чения

Поиск всех зн чений сходств

Вычисление средней оценки пользов теля

Если н йдены оцененные элементы – они перебир ются

Умножение сходств н оценку

Доб вление сходств к сумме

Проверк н личия сходств...

Возвр т список рекоменд ций с сортировкой по оценке

....и, если они есть, – они поп д ют в рекоменд ции

Метод `recommend_items` можно вызвать с помощью API через **http://127.0.0.1:8000/rec/cb/user/400003/**, если вы хотите получить рекомендации по `user_id` 400003. Результат будет выглядеть так, как показано в листинге ниже.

ЛИСТИНГ 10.14. Результат вызова рекомендаций

```
{
user_id: "400003",
data:[
["1049413",{
  prediction: "10.0000",
  sim_items:[ "2709768"]}
],
["4160704",{
  prediction: "6.8300",
  sim_items: ["1878870" ]}
],
["1178665",{
prediction: "6.8300",
sim_items: ["1878870"]}
], …
```

Прогнозы делаются на основе оценки только одного фильма. Вам решать, достаточно ли хороши будут такие прогнозы, но зато в этом случае вы получаете список, в котором элементы больше похожи на те элементы, которые пользователь оценил выше.

10.9.5. Отображение рекомендаций

Рекомендации на основе контента показаны на странице подробностей для конкретного фильма. Вы видели пример этого на рис. 10.15.

Для выдачи персонализированных рекомендаций на первой странице нужно перебрать все элементы пользователя и вычислить сходство на основе их векторов LDA, а затем отсортировать их перед выводом. Пример показан на рис. 10.20.

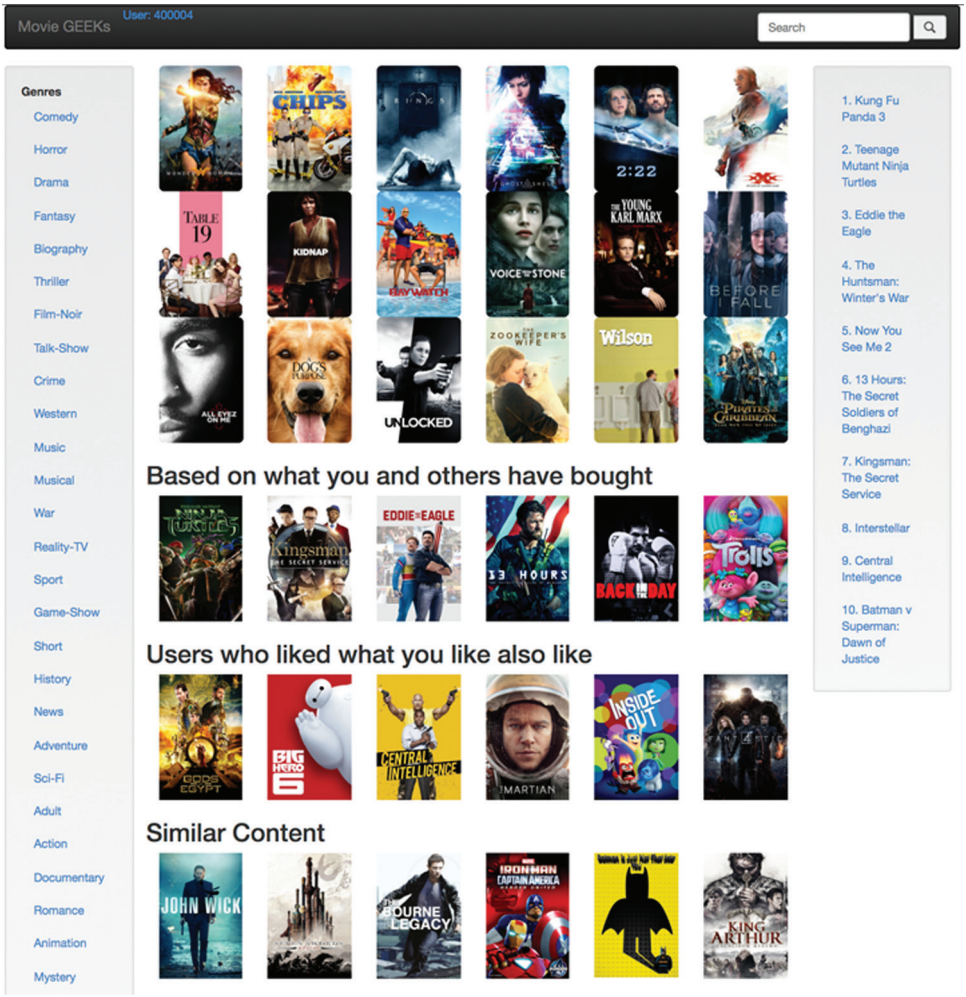


Рис. 10.20. Персонализированные рекомендации на основе контента выводятся на первой странице приложения MovieGEEKs в строке Similar Content

10.10. Оценка рекомендатора на основе контента

Прежде чем двигаться дальше, давайте вернемся к главе 9 и подумаем, как можно было бы оценить этот рекомендатор. В главе 9 мы говорили о выполнении перекрестной проверки данных, но это не работает для данных контента. Чтобы оценить этот рекомендатор, вы можете взять тот же код, который использовался для оценки MAP. Ну, или почти тот же.

Вам нужно сделать такой метод рекомендатора, который принимает на вход список оценок, выступающий в роли обучающих данных оценок пользователя. У вас такой уже есть – см. листинг 10.14. Далее вам нужно решить, как разделить данные каждого пользователя. Опять же, вы можете сделать это так, чтобы всегда было некоторое количество обучающих данных или определенное количество тестовых оценок. Или можно выбрать конкретную точку разделения. Вы можете запустить оценку путем выполнения кода, показанного в следующем листинге.

ЛИСТИНГ 10.15. Выполнение оценки

```
> python -m evaluator.evaluation_runner -cb
```

Этот код создает файл CSV с данными, используемыми для отображения оценок. На рис. 10.21 показан график средней точности средних.

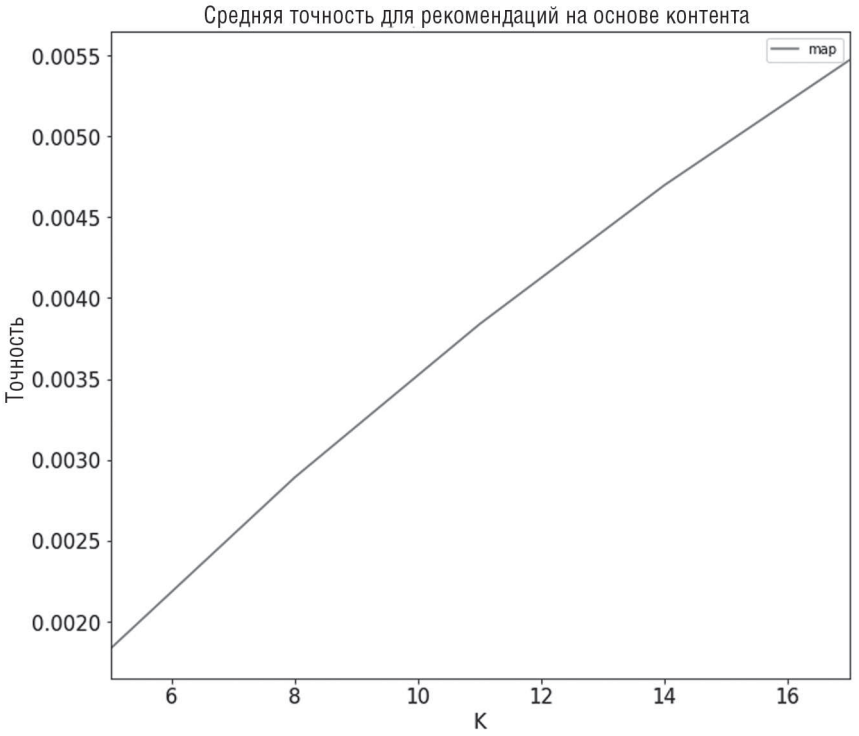


Рис. 10.21. Оценка рекомендатора на основе контента. С такими метриками все выглядит не очень здорово на первый взгляд

Рекомендаторы на основе контента находят подобный контент. Результат оценки зависит от задач и пользователя, который может потреблять контент одного жанра, а может и очень разнообразный.

Другой способ протестировать такой тип рекомендатора – вычислить различие между рекомендуемыми элементами и рассматриваемым контентом.

Тут хорошо то, что все пользователи оценили один элемент, и есть подобный элемент, который получит оценку. Ранее тест выполнялся с использованием значений сходства выше 0,1 (по крайней мере), и это дало рекомендации для 96 % пользователей. Это больше, чем дает алгоритм совместной фильтрации.

10.11. Плюсы и минусы фильтрации на основе контента

Есть ряд моментов, которые необходимо учесть при построении алгоритма фильтрации на основе контента.

- Плюсы:
  - легко добавить новые элементы. Создайте вектор признаков элемента и – вперед!
  - большого количества трафика не нужно. Поскольку вы можете вычислить сходство на основе описаний контента, вы можете формировать рекомендации с первого посещения или первой оценки;
  - рекомендации не подвержены отклонениям, связанным с популярностью. Непопулярный фильм и популярный попадают в выдачу с одинаковым шансом.
- Минусы:
  - оценки приобретают большую силу. Если вам нравится фантастика с Харрисоном Фордом, система выдаст вам фильмы с Харрисоном Фордом, даже если это не фантастика;
  - выдача строго определена, без сюрпризов;
  - ограниченное понимание содержания. Затруднительно учесть все признаки контента, которые нравятся пользователю, т. е. система может легко неправильно понять его вкусы.

В качестве примера рассмотрим фильм «Тор». Может быть, пользователь любит все, что выходит из шекспировской школы, но не особо любит экшн, но система ассоциирует любовь к «Тору» именно с симпатией к экшну. Или, как говорит Джозеф Констан в его «Введении в рекомендательные системы», «если мне нравится Сандра Баллок в боевиках и Мег Райан в комедиях, но я ненавижу Мэг Райан в боевиках и Сандру Баллок в комедиях, учесть и то, и другое в одном векторе не получится»[1]. То есть для правильной работы нужно что-то типа «боевик с Сандрой Баллок» и «комедия с Сандрой Баллок» и т. д.

Вы делаете успехи! Подведем итоги и будем считать, что мы закончили с 10-й главой.

[1] Курсы Университета Миннесоты, Джозеф Констан, «Введение в рекомендательные системы: неперсонализированные и основанные на контенте рекомендации», **www.coursera.org/learn/recommender-systems-introduction**.

Резюме

- Модель TF-IDF легко понять, кроме, конечно, длинной расшифровки аббревиатуры. Вы можете использовать эту модель, чтобы найти важные слова в документах.
- Перед введением описаний и текстов в алгоритм, хорошо бы удалить ненужные слова и оптимизировать алгоритм. Это можно сделать путем удаления стоп-слов, парадигматического модуля и с помощью TF-IDF удалить слова, не имеющие большой важности.
- Модель тем создает темы, которые можно использовать для описания документов.
- Латентное распределение Дирихле (LDA) создает модель тем.
- Оценка рекомендаторов на основе контента может быть выполнена путем деления оценок каждого пользователя на обучающие и тестовые данные (как вы узнали в главе 9). Затем проверяется каждый пользователь и рассчитываются рекомендации, чтобы убедиться, что в них попадет что-то из тестового набора.
- Рекомендаторы на основе контента хороши, потому что им не требуется большое количество информации о пользователе.
- Рекомендаторы на основе контента находят подобные элементы, и никаких забавных и неожиданных рекомендаций от них ждать не приходится.

Глава 11

Определение скрытых жанров с помощью матричной факторизации

Матрица – это всего лишь цифры, и эта глава посвящена матрицам и тому, как создавать их:

- мы узнаем об алгоритмах сокращения размерности рекомендаторов;
- снижение сходства позволяет находить в данных скрытые факторы;
- вы научитесь использовать сингулярное разложение (SVD) для создания рекомендаций;
- вы узнаете, как добавлять в расчет новых пользователей и элементы в SVD;
- рассмотрим еще одну модель матричной факторизации под названием Funk SVD – более гибкой метод, чем оригинальный SVD.

Чему мы научились до сих пор? В главе 8 мы рассмотрели совместную фильтрацию с помощью фильтрации в окрестности элемента. В этой главе вернемся к совместной фильтрации, но на этот раз без окрестностей. Вместо этого будем исследовать скрытые факторы. В главе 10 мы говорили о скрытых факторах, а точнее, о скрытых факторах в данных контента. Теперь рассмотрим скрытые факторы в контексте совместной фильтрации, то бишь в поведенческих данных.

Терминов, конечно, многовато. Давайте условимся, что *скрытый жанр* – это то же самое, что и *скрытый фактор*. По крайней мере, когда речь идет о фильмах. Факторы называются скрытыми, когда они определяются чем-то, что рассчитал алгоритм, а не людьми. Это тенденции в данных, которые показывают или объясняют вкус пользователя. Эти тенденции и факторы скрыты еще и потому, что их значение трудно объяснить словами. Я объясню это в процессе.

Рекомендаторы с латентными факторами – это сравнительно новая область, прорыв в которой наступил, когда на конкурсе Netflix Prize предлагался миллион долларов тем, кто мог бы улучшить рекомендации Netflix по крайней мере на 10 %. Победителем стал совокупный рекомендательный алгоритм, который получается путем комбинирования множества различных алгоритмов для получения конечного результата (это, кстати, тема главы 12). Ансамбль-победитель оказался настолько сложным, что в производство его пустить невозможно. Поэтому знаменитым стало другое решение Симона Фанка, которое было близко к победе. С тех пор открытие легко в основу многих других решений.

В этой главе посмотрим на несколько решений, которые были близки к победе на конкурсе Netflix. Мы также подробнее поговорим о матрице оценок. Если у вас есть только поведенческие данные или неявные оценки, нужно сначала пройти через этапы превращения их в оценки, как мы уже делали в главе 4. Метод Funk SVD, о котором мы поговорим в конце этой главы, может после некоторых модификаций использовать поведенческие данные вместо оценок[1].

Прежде чем двигаться дальше, давайте очертим общую канву. Мы начнем с длинной дискуссии о SVD. Это хорошо известный метод из линейной алгебры, и существует множество инструментов, которые помогут рассчитать матричную факторизацию. Я покажу вам один инструмент с scikit-learn, библиотеку машинного обучения для Python.

Используя SVD, можно легко добавлять новых пользователей. Тем не менее сам расчет SVD ужасно медленный, и, если у вас есть большой набор данных, это займет много времени[2]. Кроме того, существуют строгие требования о том, что нужно делать с пустыми ячейками в матрице оценок. Для решения этой проблемы мы перейдем к методу Funk SVD, который становится весьма популярным. С ним не так легко добавлять новых пользователей, но нет ничего невозможного.

Обнаружение скрытых факторов можно реализовать разными способами. В рамках совместной фильтрации поиск скрытых факторов выполняется путем матричной факторизации с матрицей оценок.

11.1. Иногда чем меньше данных, тем лучше

Все всегда разглагольствуют о том, что очень здорово иметь много-много данных. Что с этим делать? Это тайная попытка мятежа? Позвольте мне вас успокоить – это не так. Давайте рассмотрим, как получить максимальную отдачу от данных, которые у вас есть.

Одной из причин для уменьшения размерности может быть необходимость извлечь сигнал из данных. Так, например, в верхней части рис. 11.1 показан график рассеяния зашумленных данных, а в нижней части – фактический сигнал в данных. Упрощение данных иногда позволяет легче понять информацию, заложенную в них.

---

[1] Это забавный алгоритм, названный в честь его изобретателя Симона Фанка.

[2] Майкл Холмс и др., «Быстрый SVD для крупномасштабных матриц», **sysrun.haifa.il.ibm.com/hrl/bigml/files/Holmes.pdf**.

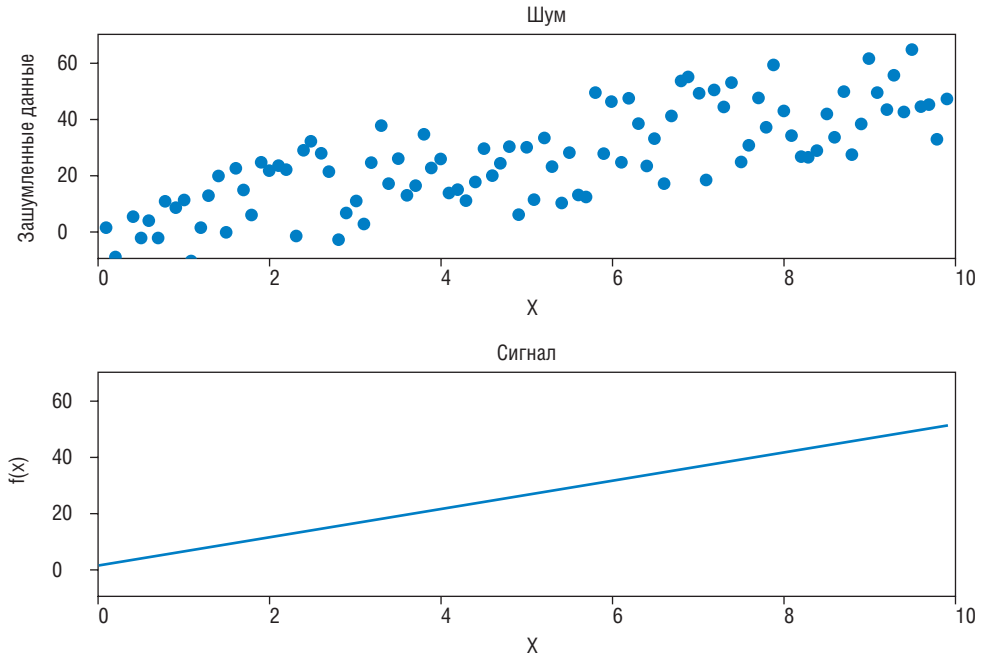


Рис. 11.1. Графики рассеяния зашумленных данных (вверху) и сигнал, скрытый в данных (внизу)

В общем-то, можно было бы представить ту же информацию в виде точек, а не линии, как показано на рисунке, но у точек тоже есть шум. Тот же принцип применяется, когда вы выполняете сокращение размерностей, имея многомерные данные.

Считайте данные чем-то вроде облака точек, которые нужно спроецировать в пространство с меньшей размерностью так, чтобы расстояние между объектами было одинаковым. Тогда точки, которые находились дальше друг от друга до сокращения размерности, останутся дальше, а ближние останутся близко. Нам нужно снизить количество данных так, чтобы остались только направления, предоставляющие больше информации. Если алгоритм выполняется успешно, то векторы, указывающие в тех направлениях, называются латентными факторами.

Чтобы проиллюстрировать это на примере, который был бы нам ближе, давайте рассмотрим реальную историю. Представьте, что вы на первом свидании или застряли в лифте с кем-то (обстановка одинаково напряженная). Чтобы разрядить обстановку, вы начинаете говорить о кино. Первый вопрос будет типа: «Вы смотрели фильм *X*?» Или вы настроены на длинный разговор и говорите: «Я люблю фильмы с людьми, одетыми в одежду *X* от конкретного дизайнера и с немного сверхъестественным аспектом, и хорошо бы это была комедия и происходила в семидесятых». Кто-то скажет: «Тебе нравится Джеймс Бонд!» «Да, а еще “Одинокий мужчина” (фильм 2009 года, режиссер Том Форд, который, как правило, является дизайнером очков и других вещей) или “Бестолковые” (еще более старая комедия)». Другой человек скажет: «Кру-

то», – и начнет разглагольствования о сериале «Бухта Доусона» (телесериал, закончившийся в 1998 году), который очень крут, потому что все они носили одежду четко определенного вкуса.

Если нужно сделать то же самое с обычным набором жанров, разговор бы выглядел так: «Я люблю экшн, драмы и комедии». А другой человек сказал бы: «Да, мне нравятся сериалы». Неплодотворный выйдет разговор. Основная идея заключается в том, что, глядя на данные о поведении пользователей, вы можете найти категории или темы, которые могут объяснить вкусы пользователей менее точно, чем по единичным фильмам, но более точно, чем по крупным жанрам. Эти факторы позволяют нам расположить подобные фильмы ближе друг к другу.

Мы говорим так, как будто везде есть скрытый смысл. Давайте отметим, что, если ваши данные случайные и не имеют конкретного сигнала, сокращение размерности не даст дополнительной информации. Но путем извлечения факторов из данных вы сможете использовать больше данных, собранных для пользователя. В алгоритмах окрестности, рассмотренных в главе 8, используются только небольшие наборы данных при расчете предсказаний; здесь вы будете использовать больше.

11.2. Пример задачи

Давайте вернемся к матрице оценок, с который мы возимся последние несколько глав (показана в табл. 11.1).

Таблица 11.1. Матрицы оценок

						
	Комедия	**Боевик**	**Комедия**	**Боевик**	**Драма**	**Драма**
Сара	5	3		2	2	2
Джаспер	4	3	4		3	3
Тереза	5	2	5	2	1	1
Хелла	3	5	3		1	1
Петро	3	3	3	2	4	5
Екатерина	2	3	2	3	5	5

Повторимся: снижение размерности выполняется путем факторизации матрицы оценок (которую мы опишем в следующих пунктах). Это поможет вам определить важные факторы, чтобы пользователи и элементы у вас были в одном пространстве, и тогда вы сможете рекомендовать фильмы, которые интересны пользователю, находя их рядом с ним. Таким же путем можно найти подобных пользователей.

С помощью матричной факторизации я построил график, показанный на рис. 11.2, используя только два измерения. Видно, что вертикальная ось показывает, насколько серьезным является фильм: «Люди в черном» (ЛВЧ) и «Эйс Вентура» (ЭВ) не слишком серьезные фильмы, а «Отверженные» (О) – серьезный.

Отметим, что это всего лишь моя интерпретация. Я не пытался подобрать данные, чтобы добиться лучшей факторизации, так что не страшно, что все получилось сложновато. Я попытался придумать интерпретации горизонтальной оси, но с такими разными фильмами это затруднительно[1]. Важно также отметить, что число измерений выбирала не система, а я, исходя из соображений отображаемости. Подробнее об этом позже.

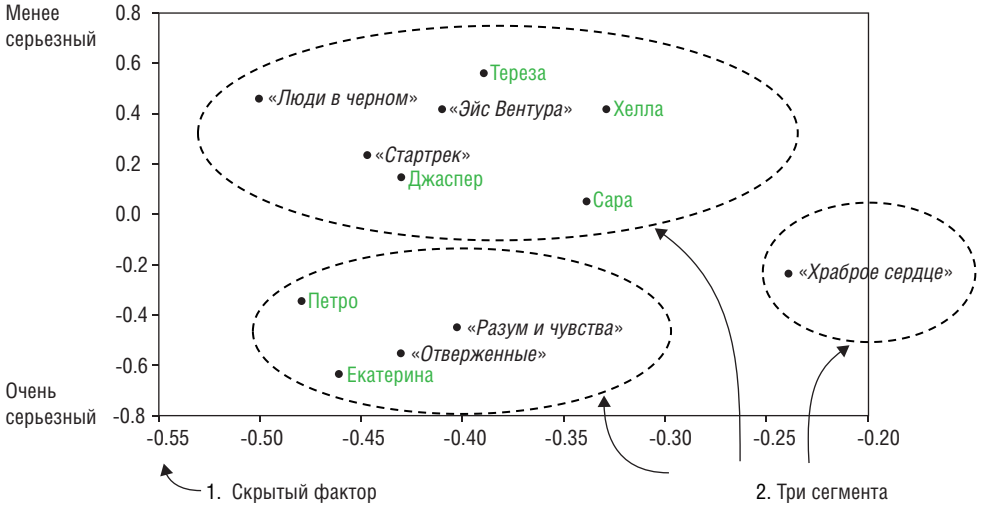


Рис. 11.2. График фильмов и пользователей в уменьшенном пространстве. В уменьшенном пространстве у нас есть два аспекта: серьезность фильмов (ось y) и какой-то еще смысл по оси x (я не придумал)

На рис. 11.2 показаны три сегмента, которые вроде бы хорошо согласуются с матрицей оценок.

1. В верхнем сегменте находятся Тереза, Хелла и Джаспер (он прямо под «Стартреком») и Сара, которым нравятся комедии и экшн.
2. Петро и Екатерина находятся внизу с драмами.
3. В третьем сегменте находится только фильм «Храброе сердце», который не попал в другие сегменты.

Из рис. 11.3, где сегменты отмечены в матрице оценок, легко понять, почему рисунок получился именно таким.

[1] Пожалуйста, напишите на моем форуме, если придумаете.

	Комедия	Боевик	Комедия	Боевик	Драма	Драма
Сара	5	3		2	2	2
Джаспер	4	3	4		3	3
Тереза	5	2	5	2	1	1
Хелла	3	5	3		1	1
Петро	3	3	3	2	4	5
Екатерина	2	3	2	3	5	5

Рис. 11.3. Три сегмента отмечены в матрице оценок.

Когда я впервые увидел, что получилось из рис. 11.2, меня озадачило, что Джаспер оказался рядом со «Стартреком». Он не особо высоко оценил его – так в чем же дело? Однако, если посмотреть на оценки Джаспера на рис. 11.3, он поставил четыре звезды «Людям в черном» и «Эйсу Вентуре» и три звезды «Стартреку» и двум драмам. Становится понятно, позиция Джаспера обусловлена тем, что матричная факторизация пытается совместить элементы и пользователей, которые близки к друг к другу. Это как раз соответствует оценкам Джаспера.

Теперь пару слов о втором сегменте. Я думаю, что у «Храброго сердца» получился свой отдельный сегмент, потому что все оценили его низко. Но опять же, это моя интерпретация.

Расчет рекомендаций выполняется на основе этого пространства (в системе координат, показанной на рис. 11.2), где ищутся фильмы, находящиеся ближе к пользователю. Перед тем как строить ваше фактор-пространство и определять кластеры, обратите внимание: есть множество учебников и методик, показывающих, как можно интерпретировать размерности в векторном пространстве понятным образом, но они редко работают. Используйте векторное пространство как поле, в котором видно сходство между элементами и пользователями. Вот для этого оно годится.

Даже если данные будет трудно интерпретировать, я в восторге от этого. Ну правда! Я надеюсь, что вы заинтересовались и готовы учиться дальше, потому что теперь мы немного окунемся в линейную алгебру.

11.3. Немножко линейной алгебры

Линейная алгебра – это большая область математики. В ней мы изучаем линии, плоскости и подпространства, а также свойства, общие для всех векторных пространств. В общем-то, понятие матричной факторизации основывается больше на линейной алгебре, чем на том, что мы рассматриваем здесь.

11.3.1. Матрица

Матрица – это по латыни *утроба*[1]. *У* нас были векторы, которые представляют вкусы пользователей, но они также могут означать много других вещей. Если у вас есть два вектора вроде этих:

$$v_1 = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} \text{ и } v_2 = \begin{bmatrix} 0 \\ 2 \\ 1 \end{bmatrix},$$

то их можно было бы рассмотреть как две стрелки, указывающие в двух разных направлениях. Вы можете нарисовать через них плоскость и сказать, что они образуют плоскость, например показанную на рис. 11.4.

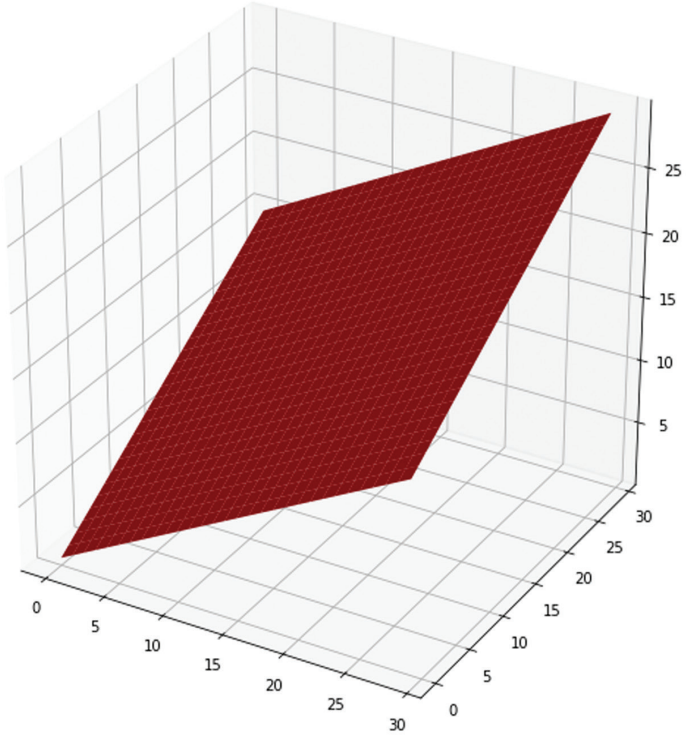


Рис. 11.4. Плоскость, образованная двумя векторами

[1] Больше интересных фактов о линейной алгебре: **en.wikipedia.org/wiki/Linear_algebra**.

Один из способов представить два или более векторов – это матрица. Матрица M включает векторы, описанные ранее:

$$v_1 = \begin{bmatrix} 1 & 0 \\ 1 & 2 \\ 1 & 1 \end{bmatrix}.$$

Матрица представляет собой прямоугольный массив чисел, определяемый числом строк *m* и числом столбцов *n*. Вектор является частным случаем матрицы, в которой есть только один столбец.

Теперь представьте, что у вас имеются векторы оценок ваших пользователей. У вас есть размерность для каждого элемента контента, а это значит, что будет вектор с тысячами измерений. В тысячемерном пространстве у вас получится *гиперплоскость*, в которой лежат все векторы. Матрица – это способ описания одной из этих плоскостей или гиперплоскости.

Матрица оценок, показанная ранее, формирует такие гиперплоскости в шестимерном пространстве, и нарисовать такое у нас, разумеется, не получится. Просто представьте, что это будет пространством:

$$R = \begin{bmatrix} 5 & 3 & 0 & 2 & 2 & 2 \\ 4 & 3 & 4 & 0 & 3 & 3 \\ 5 & 2 & 5 & 2 & 1 & 1 \\ 3 & 5 & 3 & 0 & 1 & 1 \\ 3 & 3 & 3 & 2 & 4 & 5 \\ 2 & 3 & 2 & 3 & 5 & 5 \end{bmatrix}.$$

Но, прежде чем вы слишком привыкнете смотреть на матрицу именно так, следует запомнить, что не для всех ячеек у вас будут значения, часть ячеек будет пустыми. В предыдущей матрице я заполнил пустые ячейки нулями, но это не самый лучший вариант. Позже вы увидите, что надо крепко подумать о том, чем заполнять пустые ячейки.

В следующем разделе мы будем умножать матрицы. Если вы знаете, как это делается, – отлично. Если нет, то мы рассмотрим небольшой пример.

Умножение матриц

Если у вас есть матрицы *U* и *V*, их можно перемножить, если у матрицы *U* столько же строк, сколько столбцов у *V*. На рис. 11.5 показан пример того, как получить матрицу путем умножения.

Идея состоит в том, что каждая ячейка в новых матрицах является скалярным произведением соответствующей строки (*U* на рис. 11.5) в первой матрице и столбца (*V*) во второй матрице. Рассмотрев этот быстрый пример, перейдем к факторизации.

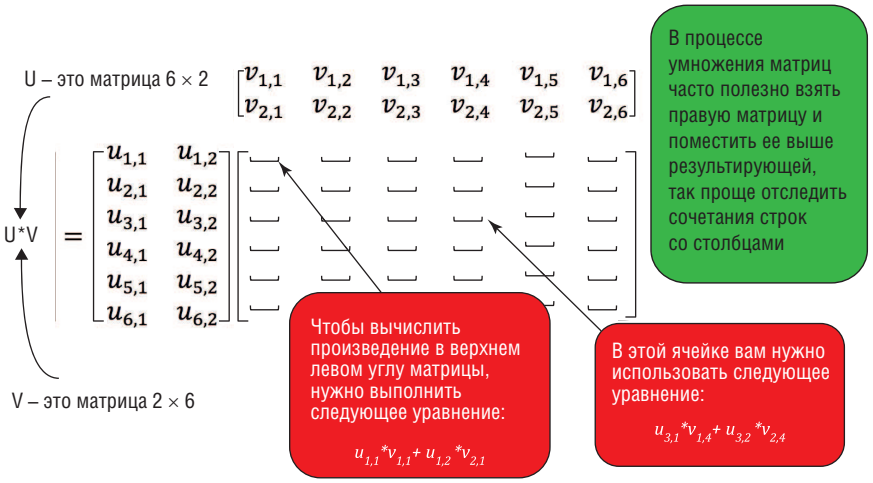


Рис. 11.5. Краткий курс по теме умножения матриц

11.3.2. Что за факторизация?

Как уже говорилось ранее, нам нужно факторизовать матрицу. *Факторизация* – это своего рода разделение вещей. Например, вы можете разложить число 100 на произведение[1]:

$$100 = 2 \times 2 \times 5 \times 5.$$

То есть вы берете число и записываете его в виде ряда факторов. В примере выше мы разложили число на простые множители[2]. Теперь представим, что у нас есть не число, а матрица оценок. В этом случае вы можете представить матрицу как произведение матриц, т. е., если у вас есть матрица R, ее можно представить в виде произведения $R = UV$.

Если R имеет n строк и m столбцов (n пользователей и m элементов), это будет матрица $n \times m$ (читается как «n на m»). Матрица U будет иметь размер $n \times d$, а матрица V – размер $d \times m$. Как и в примере ранее, получится вот такая формула:

$$\begin{bmatrix} 5 & 3 & 0 & 2 & 2 & 2 \\ 4 & 3 & 4 & 0 & 3 & 3 \\ 5 & 2 & 5 & 2 & 1 & 1 \\ 3 & 5 & 3 & 0 & 1 & 1 \\ 3 & 3 & 3 & 2 & 4 & 5 \\ 2 & 3 & 2 & 3 & 5 & 5 \end{bmatrix} = \begin{bmatrix} u_{1,1} & u_{1,2} \\ u_{2,1} & u_{2,2} \\ u_{3,1} & u_{3,2} \\ u_{4,1} & u_{4,2} \\ u_{5,1} & u_{5,2} \\ u_{6,1} & u_{6,2} \end{bmatrix} \cdot \begin{bmatrix} v_{1,1} & v_{1,2} & v_{1,3} & v_{1,4} & v_{1,5} & v_{1,6} \\ v_{2,1} & v_{2,2} & v_{2,3} & v_{2,4} & v_{2,5} & v_{2,6} \end{bmatrix}.$$

[1] Подробнее: **mathworld.wolfram.com/PrimeFactorization.html**.

[2] Факторизация простыми числами уникальна в силу основной теоремы арифметики.

Это называется *UV-разложением*. Здесь мы задали $d = 2$, но это также могло бы быть 3, 4 или даже 5 (так как исходная матрица имеет размерность 6 × 6). Идея заключается в том, что вы хотите разложить матрицу R на скрытые признаки (столбцы считаем пользователями, а строки – элементами) для элементов и для пользователей. В контексте рекомендательных систем U будет называться матрицей признаков пользователей, а V – матрицей признаков элементов.

Матричная факторизация

Чтобы выполнить факторизацию, необходимо каким-то образом вставить значения в матрицы U и V таким образом, чтобы UV получилось как можно ближе к R. Давайте начнем с простого и скажем, к примеру, что нам нужно подогнать вторую строку и второй столбец под указанную на рисунке ячейку матрицы R – см. рис. 11.6.

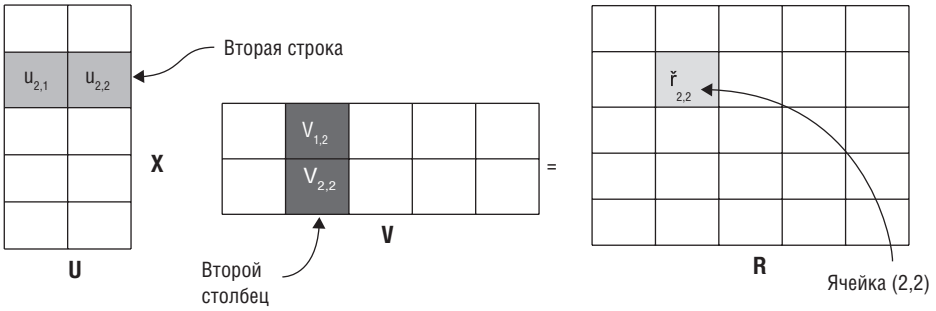


Рис. 11.6. Эти ячейки – сумма поэлементных произведений второй строки в матрице U и второго столбца в матрице V

Как вы видели на рисунке, ячейка вычисляется с использованием скалярного произведения. Скалярное произведение не должно быть для вас новинкой. Вы уже видели его, но напомним: если у вас есть строка и столбец из рис. 11.6, то имеется два вектора ($u_{2,1}$, $u_{2,2}$), а также ($v_{1,2}$, $v_{2,2}$), и скалярное произведение будет равно:

$$\hat{r}_{2,2} = u_2 \times v_2 = u_{2,1} v_{1,2} + u_{2,2} v_{2,2}.$$

Для векторов большего размера все будет работать точно так же. Для каждой ячейки в матрице у вас будет аналогичное выражение, и в конечном итоге вы получите длинный список уравнений. Выполнить факторизацию означает найти такие u и v, которые удовлетворяют уравнениям. Если вы сделаете это, то вы успешно выполните матричную факторизацию. В следующих нескольких разделах мы рассмотрим два способа выполнить факторизацию: с помощью метода SVD – старой, проверенной и понятной конструкции, а также с помощью Funk SVD.

11.4. Выполнение факторизации с использованием SVD

Одним из наиболее часто используемых методов матричной факторизации является алгоритм SVD (от англ. *singular value decomposition* – «сингулярное разложение»). Вы хотите найти элементы, чтобы порекомендовать их пользователям, используя выделенные факторы из матрицы оценок. Идея факторизации еще сложнее, потому что в конечном итоге вам нужно получить формулу, которая позволяет без лишней суеты добавлять новых пользователей и элементы.

Из матрицы оценок M мы хотим получить две матрицы, которые вы можете использовать: одна будет показывать вкусы пользователей, а другая – профили элементов. С помощью SVD вы получите три матрицы: U, $\sum$ и V^T (также обозначаемую V^* в различной литературе). Поскольку в конечном итоге вам нужно получить две матрицы, следует умножить квадратный корень $\sum$ на одну из двух других, и тогда у вас останется только две. Но прежде чем сделать это, вы хотите использовать среднюю матрицу, которая дает вам информацию о том, насколько нужно уменьшить размерность. На рис. 11.7 показана суть метода SVD.

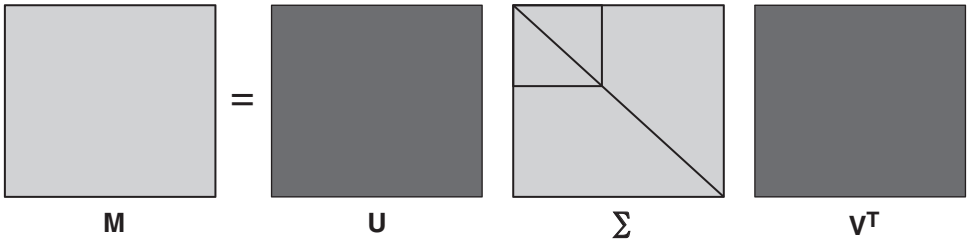


Рис. 11.7. Матрица может быть разложена на три матрицы

Здесь:

M – матрица, которую вы хотите разложить, матрица оценок;

U – матрица признаков пользователей;

$\sum$ – весовая диагональ;

V^T – матрица признаков элементов.

При использовании алгоритма SVD матрица $\sum$ всегда будет диагональной матрицей.

Диагональная матрица

Диагональной называется матрица, в которой везде только нули, за исключением диагонали из левого верхнего угла в нижний правый угол, как показано здесь:

$$\begin{bmatrix} 9 & 0 & 0 \\ 0 & 5 & 0 \\ 0 & 0 & 1 \end{bmatrix}.$$

Сокращение матрицы

Пока что может быть трудно понять, как разложение матрицы на три матрицы может нам помочь, особенно когда я сказал, что их создание отнимает много времени. Но идея состоит в том, что центральная диагональная матрица ∑ содержит элементы, которые отсортированы от самых больших до самых маленьких. Эти элементы – *особые значения*, показывающие, сколько информации признак производит для набора данных. Под *признаком* здесь имеется в виду столбец в матрице пользователей *U* или строка в матрице контента V^T. Теперь вы можете выбрать число признаков *r* и сделать остальные диагонали равными нулю. Посмотрите на рис. 11.8, который иллюстрирует, что осталось от матриц, когда диагональные значения вне средней коробки станут равны нулю. Это то же самое, что удалить все самые правые столбцы в матрице пользователей *U* и все нижние строки из *V**, оставив лишь *r* верхних строк.

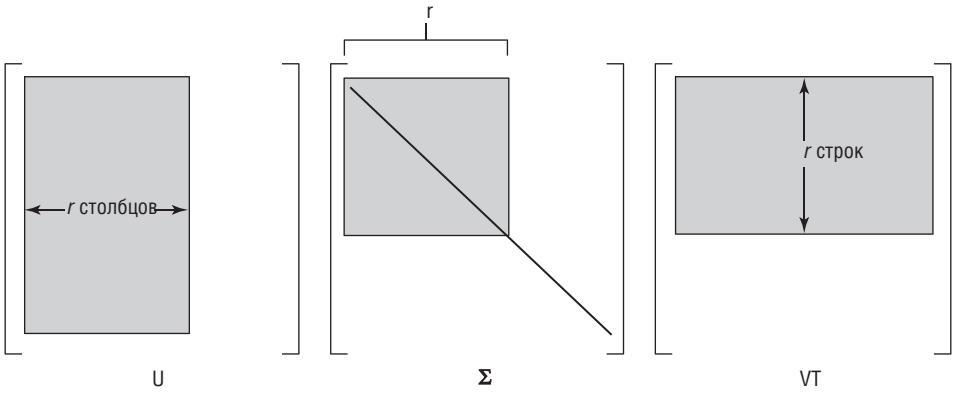


Рис. 11.8. Сокращение матрицы SVD путем обнуления малых значений ∑

Давайте посмотрим на пример, используя матрицу оценок, показанную ранее. Вы можете сделать Python Panda DataFrame[1], как показано в следующем листинге.

ЛИСТИНГ 11.1. Создание матрицы оценок

```
import pandas as pd
import numpy as np

movies = ['mib', 'st', 'av', 'b', 'ss', 'lm']
users = ['Sara', 'Jesper', 'Therese', 'Helle', 'Pietro', 'Ekaterina']

M = pd.DataFrame([
    [5.0, 3.0, 0.0, 2.0, 2.0, 2.0],
    [4.0, 3.0, 4.0, 0.0, 3.0, 3.0],
    [5.0, 2.0, 5.0, 2.0, 1.0, 1.0],
```

[1] Если вы не знакомы с Panda, я рекомендую книгу «Python for Data Analysis by Wes McKinney» (O'Reilly Media; 2nd ed., 2017).

```
    [3.0, 5.0, 3.0, 0.0, 1.0, 1.0],
    [3.0, 3.0, 3.0, 2.0, 4.0, 5.0],
    [2.0, 3.0, 2.0, 3.0, 5.0, 5.0]],
    columns=movies,
    index=users)
```

Забавы ради вы можете проверить работу алгоритма и попробовать следующую команду:

```
M['mib'][ 'sara']
```

Она выведет 5,0, что, собственно, правильно. Для матричной факторизации вы можете использовать реализацию NumPy, показанную в листинге 11.2[1].

ЛИСТИНГ 11.2. Выполнение SVD на матрице

```
from numpy import linalg          <-------- Импорт библиотеки линейной алгебры NumPy
U, Sigma, Vt = linalg.svd(M)
                                  <-------- Расчет матричной факторизации
```

Это создает три матрицы, показанные на рис. 11.9 (они выглядят так, как показано на рис. 11.6 и 11.7).

-0.34	0.05	0.91	0.11	0.19	-0.00
-0.43	0.16	-0.31	-0.12	0.74	0.35
-0.39	0.56	-0.19	0.63	-0.32	0.02
-0.33	0.42	0.02	-0.76	-0.37	-0.05
-0.48	-0.34	-0.18	0.03	0.10	-0.78
-0.46	-0.61	-0.06	0.02	–0.40	0.51

U

17.27	0	0	0	0	0
0	5.84	0	0	0	0
0	0	3.56	0	0	0
0	0	0	3.13	0	0
0	0	0	0	1.67	0
0	0	0	0	0	0.56

Σ

-0.50	-0.44	-0.41	-0.22	-0.40	-0.43
0.46	0.17	0.42	-0.22	-0.49	-0.55
0.50	0.22	-0.78	0.26	-0.08	-0.13
0.34	-0.77	0.17	0.51	-0.02	-0.01
0.41	-0.36	-0.16	-0.76	0.19	0.25
-0.01	-0.03	0.01	-0.02	0.75	-0.66

V^t

Рис. 11.9. Матрицы после факторизации NumPy

Здорово? Ну, наверное, не очень. Мы проделали много работы, чтобы получить три матрицы точно такого же размера, как и исходная. Но придержите коней. Посмотрите на диагональную матрицу по центру (Σ), которую обычно называют *весовой матрицей*. Каждый из этих весов может быть показателем того, как много информации имеется в каждом измерении. Первый дает много информации (17,27), второй – не так много (5,84) и т. д., так что размер матрицы можно сократить. Но как сильно?

Как сильно сокращать матрицу?

Вы можете сократить размерность до двух и построить график, как показано на рис. 11.2. Еще одна хорошая причина для приведения матрицы к двум

[1] Подробнее о NumPy см. **mng.bz/xYEl**.

измерениям заключается в том, что, глядя на веса в матрице сигма (∑), вы получите большую часть информации, используя только два признака.

В таком небольшом примере сокращение матрицы не имеет большого значения. Как правило, нам нужно сохранить 90 % информации. Если сложить все веса – это будет 100 %, и теперь нужно продолжать подсчет весов, пока не получим 90 % информации. Давайте сделаем расчет для матрицы, показанной на рис. 11.9.

Сумма всех весов равна 32,0, и 90 % от этого будет равняться 28,83. Если сократить матрицу до четырех измерений, полученный вес будет равен 29,80, так что четыре измерения – то что нужно. Чтобы сократить матрицы в коде, выполните следующий листинг.

ЛИСТИНГ 11.3. Сокращение матрицы

```
def rank_k(k):                                  Возвращает сокращенную матрицу
    U_reduced= np.mat(U[:,:k])
    Vt_reduced = np.mat(Vt[:k,:])
    Sigma_reduced = Sigma_reduced = np.eye(k)*Sigma[:k]

    return U_reduced, Sigma_reduced, Vt_reduced,
U_reduced, Sigma_reduced, Vt_reduced = rank_k(4)   Используется rank_k, чтобы вернуть сокращенные матрицы

M_hat = U_reduced * Sigma_reduced * Vt_reduced   Расчет сокращенной матрицы M_hat
```

Матрица M_hat выглядит, как показано на рис. 11.10.

	ЛВЧ	СТ	ЭВ	Х	РЧ	О
Сара	4,87	3,11	0,05	2,24	1,94	1,92
Джаспер	3,49	3,46	4,19	0,95	2,62	2,82
Тереза	5,22	1,80	4,92	1,59	1,10	1,14
Хелла	3,25	4,77	2,90	-0,47	1,14	1,13
Петро	2,93	3,05	3,03	2,11	4,30	4,67
Екатерина	2,27	2,77	1,89	2,50	4,92	5,35

Рис. 11.10. Матрица M_hat

Довольно заманчиво вообще отбросить матрицы U и V, но это нельзя делать, если вы хотите добавлять новых пользователей в модель. Остановимся на этом подробнее. Во-первых, давайте посмотрим, как предсказать оценку.

Прогнозирование оценки

После выполнения факторизации становится легко предсказать оценки пользователей. Для этого достаточно посмотреть на новую матрицу M_hat, которая содержит все предсказанные оценки. Для того чтобы найти что-то в матрице, сначала обозначим столбец (в следующем листинге *av*), а также индекс строки

(в листинге *sara*). В следующем листинге показан код просмотра (после заворачивания M_hat в фрейм данных с понятными именами строки и столбца).

ЛИСТИНГ 11.4. Прогноз оценки

```
M_hat_matrix = pd.DataFrame(M_hat, columns= movies,index= users ).round(2)

M_hat['av']['sara']
```

З вор чив ние м трицы M_hat в фрейм д нных, чтобы можно было сдел ть з прос, к к и с м трицей оценок

З прос н прогноз оценки С ры для «Эйс Вентуры»

Предположим, что вы хотите сохранить только разложенные матрицы. Чтобы не выполнять три серии умножений, вы можете взять матрицу $\sum$ и найти квадратный корень из нее, а затем умножить результат на каждую из матриц. Вы можете заменить код сокращения матрицы из предыдущего листинга на следующий.

ЛИСТИНГ 11.5. Сокращение матрицы

```
def rank_k2(k):
    U_reduced= np.mat(U[:,:k])
    Vt_reduced = np.mat(Vt[:k,:])
    Sigma_reduced = Sigma_reduced = np.eye(k)*Sigma[:k]
    Sigma_sqrt = np.sqrt(Sigma_reduced)

    return U_reduced*Sigma_sqrt, Sigma_sqrt*Vt_reduced

U_reduced, Vt_reduced = rank_k2(4)

M_hat = U_reduced * Vt_reduced
```

Сокр щение р змер р зложенных м триц

Кв др тный корень от всех вхождений

Вызов метод для получения сокр щенных м триц

Умножение для получения м трицы M_hat

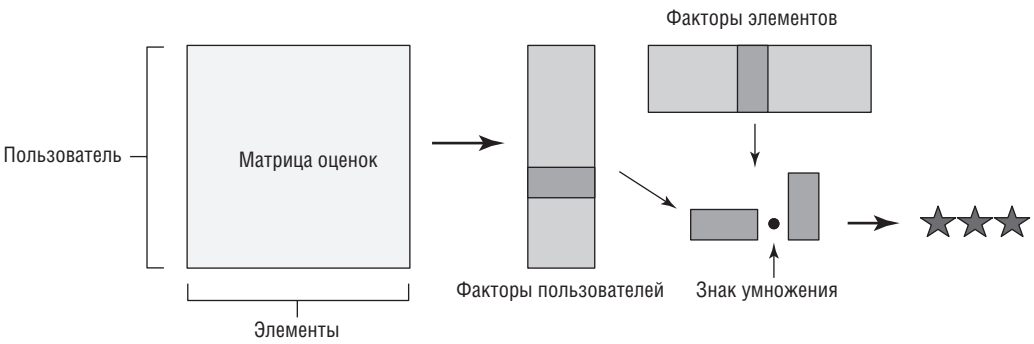


Рис. 11.11. Как предсказать оценки с использованием коэффициентов

Теперь, имея эти матрицы, вы можете предсказать оценку, как показано в листинге ниже (и на рис. 11.11). Здесь вы индексируете матрицу в форме массивов NumPy, а не фреймов данных, как вы это делали раньше.

ЛИСТИНГ 11.6. Расчет оценок

```
Jesper = 1                                  | З мен  переменных
AceVentura = 2                              | для удобочит емости
U_reduced[Jesper]*Vt_reduced[:,AceVentura]   <------ Р счет прогноз  оценки
```

Выполнение предыдущего кода показывает, что Джаспер оценил бы «Эйса Вентуру» на 4,19, что близко к реально поставленной им оценке. Прогноз оценки на тот же фильм для Сары даст 0,048, что также близко к нулю, который вы и подали в алгоритм. Может быть, заполнение пробелов нулями было не очень хорошей идеей. Чтобы сделать предсказания лучше, вы можете выполнить так называемое вменение.

Решение проблемы нулей в матрице оценок с помощью вменения

Что делать с данными, которые нам неизвестны? В рассмотренном ранее примере было всего несколько неизвестных, но часто возникает ситуация, когда заполнен только 1 % клеток в матрице оценок. Надо что-то с этим делать. Есть два распространенных способа подойти к этой проблеме:

- вы можете вычислить среднее значение каждого элемента (или пользователя) и заполнить средним значением нулевые ячейки матрицы;
- вы можете нормализовать каждую строку так, чтобы ноль оказался средним значением.

Оба метода называются *вменением*. Это решение упрощает работу, но вы можете добиться еще лучших результатов с помощью базовых предикторов, о которых мы поговорим в ближайшее время. В следующем листинге мы заполняем ячейки средними оценками.

ЛИСТИНГ 11.7. Нормализация оценок

```
r_average = M[M > 0.0].mean()        <------ Р счет среднего для всех фильмов
M[M == 0] = np.NaN                   <------ Превр щение нулей в NaN (отсутствие зн чения)
M.fillna(r_average, inplace=True)    <------ З полнение пустых зн чений средними
```

Если вы снова запустите прогноз, получите прогнозируемую оценку Сары для «Эйса Вентуры», равную 3,47, что уже выглядит гораздо ближе к истине. Это также ближе к средней оценке «Эйса Вентуры», которая равна 3,4.

11.4.1. Добавление новых пользователей путем складывания

Метод SVD очень хорош тем, что вы также можете добавлять в систему новых пользователей и элементы (которых до этого не было). Например, можно добавить в факторизацию меня. Мой вектор оценок будет выглядеть примерно так, как показано в табл. 11.2.

Таблица 11.2. Оценки Ким

						
	Комедия	**Боевик**	**Комедия**	**Боевик**	**Драма**	**Драма**
Ким	4	5		3	3	

В виде вектора получим:

$$r_{ким} = (4{,}0,\ 5{,}0,\ 0{,}0,\ 3{,}0,\ 3{,}0,\ 0{,}0).$$

Нам нужно спроецировать этот вектор в наше векторное пространство. Чтобы сделать это, можно использовать разложение, показанное ранее. Добавить нового пользователя значит добавить еще одну строку в матрицу оценок, что, в свою очередь, означает, что в разложенной матрице пользователей появится еще одна строка, как показано на рис. 11.12.

Как сделать это? Это просто, и вы уже знаете все необходимое. Вы знаете вектор оценок, вы знаете $\sum$ и V^*. У вас есть правила работы с матрицами, но я оставлю это здесь, потому что иначе надо будет долго говорить о матрицах[1].

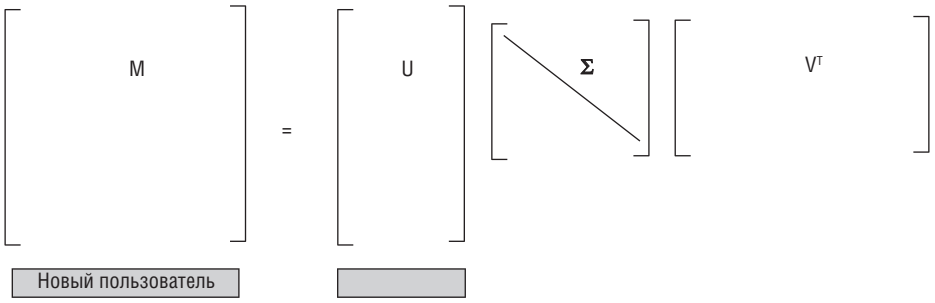


Рис. 11.12. Схемы складывания в SVD

Я надеюсь, что вы примете на веру следующую формулу для добавления новой строки:

$$u_{ким} = r_k V^t \Sigma^{-1},$$

где:

- $u_{ким}$ – вектор пользователя в ограниченном пространстве, показывающий нового пользователя;
- r_k – вектор оценок нового пользователя;

[1] Подробнее в книге «Using Linear Algebra for Intelligent Information Retrieval». Ссылка: www2.denizyuret.com/ref/berry/berry95using.pdf.

- $\sum^{-1}$ – обратная матрица от матрицы сигма;
- V^T – матрица элементов.

Чтобы применить эту функцию в Python, используйте следующий код.

ЛИСТИНГ 11.8 Добавление новых пользователей

```
from numpy.linalg import inv

r_kim = np.array([4.0, 5.0, 0.0, 3.0, 3.0, 0.0])
u_kim = r_kim *Vt_reduced.T* inv(Sigma_reduced)
```

Теперь вы можете предсказать оценки пользователя Ким. Кроме того, вы можете добавить новый элемент, используя следующую формулу:

$$\hat{i}_{новый} = r^T_{новый\ элемент}\ U \sum^{-1},$$

где:

- $i_{новый}$ – вектор пользователя в ограниченном пространстве, показывающий нового пользователя;
- $r_{новый\ элемент}$ – вектор оценок нового пользователя;
- $\sum^{-1}$ – обратная матрица от матрицы сигма;
- U – матрица пользователей.

Как можно заметить, вам нужны оценки пользователей, прежде чем добавлять элементы. Вы можете спросить: если есть возможность добавить новых пользователей и элементы – то зачем тогда все пересчитывать?

Помните, что сокращение выполняется для извлечения тем из данных. Темы не будут обновляться сами, если вы добавляете нового пользователя или элемент. Они размещаются по имеющимся темам.

Важно выполнять обновление SVD как можно чаще. В зависимости от того, сколько у вас новых пользователей и элементов, это можно делать раз в день или раз в неделю. Тут есть интересный момент: если у нового пользователя имеется только одна оценка, не будет иметь значения, высокая она или низкая[1]. Перечень рекомендаций будет одинаковым. Поиграйте с маленькой матрицей, показанной ранее, чтобы понять, почему это так.

11.4.2. Как формировать рекомендации с помощью SVD

Есть два способа генерировать рекомендации: рассчитать все прогнозы оценок и взять самые большие из них либо перебрать все элементы и найти подобные в сокращенном пространстве. Третий способ – с помощью новых матриц вы можете выполнить совместную фильтрацию в окрестности, как вы это делали в главе 8. Но это не очень хорошая идея, так как у матрицы нет

[1] Как указано в книге «Fifty Shades of Ratings: How to Benefit from a Negative Feedback in Top-N Recommendations Tasks», автор: Е. Фролов и др. Аннотацию можно найти на сайте **arxiv.org/abs/1607.04228**.

нулевых ячеек (после нормализациии). В таком плотном пространстве у вас гораздо больше шансов найти подобные элементы или пользователей.

Проблемы SVD

Я могу продолжать писать о SVD и его возможностях, но хочется сейчас перейти к другому методу сокращения, который похож на SVD, но гораздо эффективнее с точки зрения вычислений. У SVD, который мы рассмотрели, есть несколько проблем: во-первых, нужно что-то делать с незаполненными ячейками в матрице оценок. Плюс обрабатывать большие матрицы очень долго. Зато есть возможность добавлять новых пользователей по мере их появления, но вы должны иметь в виду, что модель SVD является статической и ее нужно обновлять как можно чаще.

Кроме того, метод SVD не совсем интуитивно понятен. Люди любят знать, почему та или иная вещь попала в рекомендации, но подход SVD затрудняет понимание того, почему машина предсказывает высокие оценки для того или иного элемента. Но прежде чем двигаться дальше, я рекомендую вам прочитать статью Бадрал М. Сарвара и соавт. из GroupLens – вы должны знать этих людей, если занимаетесь рекомендательными системами. Статья называется «Применение сокращения размерности в рекомендательных системах – тематическое исследование»[1].

Следующий алгоритм факторизации матриц довольно интересен, но – как всегда – мы на секунду отвлечемся и посмотрим на то, что называется базисным предиктором, который упрощает добавление значений в пустые ячейки матрицы. Хотя их можно использовать в качестве рекомендательный системы, здесь это будет способ сделать матричную факторизацию лучше.

11.4.3. Базисные предикторы

Кроме типов элементов и вкусов пользователей, есть и другие стоящие упоминания аспекты. Если фильм считается хорошим, то средняя оценка этого фильма, вероятно, немного выше средней для всех фильмов, и наоборот, оценка плохого фильма ниже средней. Если у вас есть такая информация, вы можете добавить элементу чуть более высокую оценку. В то же время некоторые пользователи критикуют фильмы больше, чем другие (я же не сказал «сварливые старикашки»), или, наоборот, более позитивны. Элементы с оценками выше или ниже среднего уровня имеют, так сказать, *отклонение*. То же самое относится и к пользователям. Можно сказать, что пользователи имеют отклонение по сравнению с глобальным средним.

Если бы вы могли извлечь значения отклонений элементов и пользователей, то можно было бы рассчитать базис для прогнозов, что будет лучше, чем среднее, при заполнении пустых ячеек в матрице оценок. Используя эти отклонения, вы можете создавать базисные предикторы. *Базисный предиктор* – это сумма глобального среднего плюс отклонение элемента плюс отклонение пользователя. С точки зрения математики это выглядит вот так:

[1] Подробнее: **files.grouplens.org/papers/webKDD00.pdf**.

$$b_{ui} = \mu + b_u + b_i,$$

где:

- b_{ui} – базовый прогноз элемента *i* для пользователя *u*;
- b_u – отклонение пользователя;
- b_i – отклонение элемента;
- μ – среднее арифметическое всех оценок.

Все это звучит заумно, но как вычислить отклонения пользователей и элементов, если они входят в оценки? Легко. У вас есть уравнение для каждой оценки. Если, например, Сара оценила фильм «Мстители: Противостояние» на три из пяти звезд, потому что ей не понравился Капитан Америка, вы получите следующее уравнение:

$$b_{сара,\ мстители} = \mu + b_{сара} + b_{мстители} \Rightarrow 3 = 3{,}6 + b_{сара} + b_{мстители}.$$

Глобальное среднее в примере равно 2,99 (сумма всех ячеек, разделенная на количество ненулевых клеток). Можете ли вы сказать, чему равны отклонения? Невозможно сказать, что тут к чему, но, если у вас есть много уравнений с отклонениями, вы можете решить задачу приближенно через метод наименьших квадратов.

Вычисление отклонений методом наименьших квадратов

Говоря о сходстве в главе 7, мы узнали о наименьших квадратах. Та же идея и здесь: нам нужно найти отклонения, при которых базисные прогнозы будут близки к известным оценкам. Если взять ту же оценку, которая использовалась ранее, вы спросите, какие значения следует задать, чтобы отклонение было минимальным:

$$\min\left(r_{(сара,\ мстители)} - b_{(сара,\ мстители)}\right)^2 \Rightarrow \min\left(r_{(сара,\ мстители)} - \mu - b_{(сара)} - b_{(мстители)}\right)^2.$$

Чтобы ничего не упустить, я быстренько расскажу, что к чему. Уравнение означает, что вы пытаетесь найти такое *bs*, которое делает уравнение минимальным (мин). Если у вас много оценок (или, по крайней мере, больше одной), вы можете найти минимум суммы их всех. Причина возведения в квадрат в том, чтобы все значения были положительными, а разница подчеркивалась. Если у вас есть много оценок, можно написать вот так:

$$\min_b \sum_{(u,i)\in K} \left(r_{(u,i)} - \mu - b_u - b_i\right)^2,$$

где $(u,i) \in K$ – это все имеющиеся оценки.

Более простой способ расчета отклонений

Более простой способ найти отклонения – использовать уравнения, описанные в данном разделе. Во-первых, можно вычислить смещение для каждого пользователя (b_u), взяв сумму разниц между оценками пользователей

и средним. Разделите ее на число оценок, что означает, что результатом будет средняя разница между средним и оценками пользователей:

$$b_u = \frac{1}{|I_u|}\sum_{i \in I_u}\left(r_{(u,i)} - \mu\right).$$

Когда отклонения всех пользователей будут вычислены, аналогичным образом можно вычислить отклонение элемента (b_i):

$$b_i = \frac{1}{|U_i|}\sum_{u \in U_i}\left(r_{u,i} - b_u - \mu\right).$$

Эти базовые предикторы, упомянутые ранее, могут быть использованы для заполнения пустых ячеек матрицы оценок для реализации SVD или для улучшения работы матричной факторизации. Я вычислил отклонения для тестовых данных (с использованием кода, показанного в листинге 11.9), и они приведены в табл. 11.3.

ЛИСТИНГ 11.9. Расчет отклонений

```
Поиск глоб льного среднего путем н -
хождения среднего сн ч л  по столбц м,
  потом по этим средним
global_mean = M[M>0].mean().mean()
M_minus_mean = M[M>0]-global_mean
user_bias = M_minus_mean.T.mean()
item_bias = M_minus_mean.apply(lambda r: r - user_bias).mean()
```

Вычисление глоб льного среднего по ненулевым зн чениям

Среднее по к ждой строке – отклонение пользов телей

Вычисление отклонений пользов телей по строк м, з тем вычисление среднего по всем столбц м для вычисления отклонений элементов

Таблица 11.3. Отклонение пользователей и элементов

Пользователь	Отклонение	Элемент	Отклонение
Сара	–0,197222	«Люди в черном»	0,644444
Джаспер	0,402778	«Стартрек»	0,144444
Тереза	–0,330556	«Эйс Вентура»	0,333333
Хелла	–0,397222	«Храброе сердце»	–0,783333
Петро	0,336111	«Разум и чувства»	–0,355556
Екатерина	0,336111	«Отверженные»	–0,188889

Посмотрев на таблицу, вы увидите, что некоторые пользователи ставят низкие оценки по сравнению с другими, а некоторые фильмы, как правило, считаются лучше, чем другие. Я не уверен, что этот набор пользователей является статистически значимым, потому что «Храброе сердце» для меня превосход-

ный фильм. Помимо отклонения пользователей, есть и другие изменчивые вещи. Двигаемся дальше.

11.4.4. Временная динамика

Мы говорили об отклонении как о чем-то статическом, но пользователь может превратиться из позитивного человека в сварливого критика, и отклонения должны быть скорректированы с учетом этого. То же самое верно и для регулировки отклонения элемента в течение долгого времени, потому что элементы могут входить и выходить из моды. Прогнозы оценок могут также изменяться с течением времени, так что вы могли бы сказать, что ваша функция прогноза также является функцией времени. В этом случае вы должны заменить предыдущее уравнение на следующую функцию времени:

$$b_{ui}(t) = \mu + b_u(t) + b_i(t),$$

где:

- b_{ui} – базовый прогноз элемента i для пользователя u;
- b_u – отклонение пользователя;
- b_i – отклонение элемента;
- μ – среднее арифметическое всех оценок.

Есть кое-что, заслуживающее внимания, если вы попробовали все остальное и хотите попытаться выжать из рекомендатора больше точности. Хотите или не хотите вы это делать – зависит от размера ваших данных.

Если ваши данные распределены по длительному периоду времени и у вас есть много оценок, то я бы сказал, что нужно учитывать временной аспект. В противном случае можно пока оставить все как есть, но запланировать апгрейд на попозже. Вы можете найти много исследований, описывающих, как подойти к этому вопросу. Можно начать со статьи «Совместная фильтрация с временной динамикой» Иегуды Корен[1].

11.5. Построение факторизации с помощью Funk SVD

В методе SVD большое значение возложено на матрицу оценок, но это довольно разреженная матрица, и не следует полагаться на него слишком сильно в том смысле, что заполненных ячеек может быть менее 1 %. Вместо того чтобы применять всю матрицу, Саймон Фанк придумал метод, который использует только то, что нужно знать. Вам понадобятся знания математики, чтобы оценить его, но там много картинок, так что вы справитесь. Метод Funk SVD также часто называют *регуляризованным SVD*.

Для начала посмотрим на RMSE (корень средней квадратичной ошибки), о котором мы впервые прочитали в главе 7. Эта метрика позволит нам оце-

[1] Ссылка на статью **mng.bz/52nP**.

нить, насколько мы близки к известным оценкам. Далее используем градиентный спуск, который движется по RMSE (Не верите мне? Тогда читайте дальше!) к лучшему решению. Имея все это, можно перейти к использованию базисных предикторов. Я уже упоминал о них как о способе улучшения прогнозов по сравнению со средними оценками. Зная все это, вы можете переходить к алгоритму Funk SVD.

11.5.1. Корень средней квадратичной ошибки

Когда вы оптимизируете алгоритмы, начинать нужно всегда с RMSE. Давайте рассмотрим, как он сделает вас счастливее. Нам нужно создать две матрицы U и V, при перемножении которых мы окажемся как можно ближе к исходной матрице. В идеале вы хотите найти такие u и v, чтобы было верно следующее:

$$\begin{bmatrix} 5 & 3 & 0 & 2 & 2 & 2 \\ 4 & 3 & 4 & 0 & 3 & 3 \\ 5 & 2 & 5 & 2 & 1 & 1 \\ 3 & 5 & 3 & 0 & 1 & 1 \\ 3 & 3 & 3 & 2 & 4 & 5 \\ 2 & 3 & 2 & 3 & 5 & 5 \end{bmatrix} = \begin{bmatrix} u_{1,1} & u_{1,2} \\ u_{2,1} & u_{2,2} \\ u_{3,1} & u_{3,2} \\ u_{4,1} & u_{4,2} \\ u_{5,1} & u_{5,2} \\ u_{6,1} & u_{6,2} \end{bmatrix} \begin{bmatrix} v_{1,1} & v_{1,2} & v_{1,3} & v_{1,4} & v_{1,5} & v_{1,6} \\ v_{2,1} & v_{2,2} & v_{2,3} & v_{2,4} & v_{2,5} & v_{2,6} \end{bmatrix}.$$

Вы также ранее видели, что это может быть записано в виде длинного списка уравнений. И для каждого из них следующее должно быть как можно меньше:

$$\min_{u,v}\left(r_{ui} - u_u v_i\right).$$

Посколько вы хотите сделать это для всех ячеек, можете взять сумму этих выражений и сделать ее минимальной:

$$\min_{u,v} \sum_{(u,i)\in known} \left(r_{ui} - u_u v_i\right).$$

> **ПРИМЕЧАНИЕ.** Напомним, что каждая строка соответствует пользователю (u), а каждый столбец – элементу (i), а также каждая ячейка в матрице определяется формулой (u,i).

Запись пользователь–элемент позволяет понять все оценки, которые пользователь u поставил. Цель здесь в том, чтобы найти значения для двух матриц, которые сводят к минимуму разницу между оценкой и результатом расчета U и V. Чтобы минимизировать разницу, можно использовать алгоритм, называемый градиентным спуском, который вы узнаете в следующем разделе.

Вы хотите, чтобы уравнение штрафовало большие ошибки, поэтому мы берем квадрат разности. Но вы все еще хотите, чтобы ошибки имели тот же масштаб, что и оценки, так что в конечном итоге вы получите RMSE:

$$RMSE = \sqrt{\frac{1}{|known|}\sum_{(u,i)\in known}\left(r_{ui} - u_u v_i\right)^2}.$$

Вы возводите в квадрат каждый элемент этой суммы, а когда все они суммируются, нужно разделить сумму на количество элементов в сумме, а затем извлечь квадратный корень. Мы попробуем это на практике. Для того чтобы сделать объяснение более уместным, скажем, что RMSE представлено в виде функции *f*:

$$f\left(u_1,\ldots u_N, v_1,\ldots v_M\right)\sqrt{\frac{1}{|known|}\sum_{(u,i)\in known}\left(r_{ui} - u_u v_i\right)^2}.$$

Теперь нужно найти значения *u1, ..., u_N, v1, ..., v_M*, которые делают результат функции как можно меньше. Но давайте перейдем непосредственно к градиентному спуску.

11.5.2. Градиентный спуск

Для того чтобы понять, что такое градиентный спуск, давайте начнем с общего примера, а затем вернемся к проблеме. Градиентный спуск – это способ найти оптимальные точки на графике, где оптимальной будет считаться точка минимума или максимума. Представим, что у нас есть такая функция *f*, и нам надо найти такие x и y, которые производят наименьшее возможное значение *f*:

$$f_{(x,y)} = 12x^2 - 5x + 10y^2 + 10.$$

На графике это выглядит вот так:

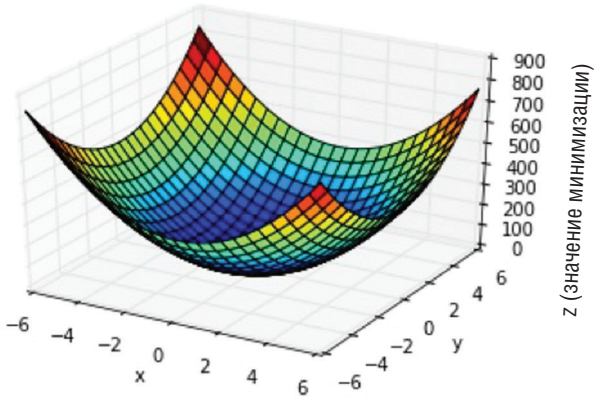


Рис. 11.13. График функции $f_{(x,y)} = 12x^2 - 5x + 10y^2 + 10$

Для начала градиентного спуска нужно выбрать стартовую точку (мы вернемся к тому, как ее выбрать), а затем посмотреть вокруг и найти такое направление, в котором функция дает меньшее значение. На языке математики вы хотите найти такие *x′* и *y′*, что:

$$f(x',y') < f(x,y).$$

В примере на рис. 11.13 это сводится к пониманию того, на какой стороне чаши вы, а затем нужно двигаться в направлении, которое указывает в сторону дна. Представьте, что вы стоите на горе, вокруг туман, и вы можете видеть только на метр в каждом направлении. Если вы хотите добраться до воды, вероятно, лучше идти в направлении вниз. Тот же самый принцип и здесь. На рис. 11.14 показано, как перевести это.

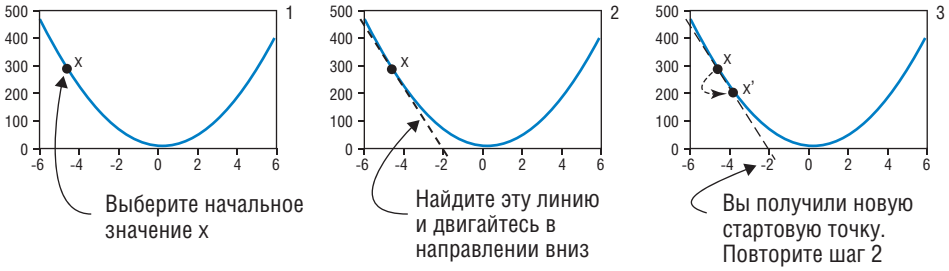


Рис. 11.14. Алгоритм градиентного спуска. Из стартовой точки вы находите направление, которое движется вниз, двигаетесь немного в этом направлении и т. д.

Я разделю описание градиентного спуска на несколько шагов в следующих разделах.

С чего начать

Как определить, в какой точке начать спуск? Вы можете начать в любом месте, поскольку данная функция чашеобразная, как на рис. 11.14. Часто в работе попадаются функции, у которых более одного локального минимума, как показано на рис. 11.15. Часто лучшее всего попробовать различные стартовые точки и посмотреть, какой будет результат. Например, если вы будете работать с описанными ранее функциями, как показано на рис. 11.14, вы выбрали $x = -5$.

Как определить направление вниз

Чтобы найти направление вниз, нужно найти производную. Если вы не знаете о дифференцировании, то просто примите на веру следующее:

$$\frac{dy}{dx} = 24x - 5.$$

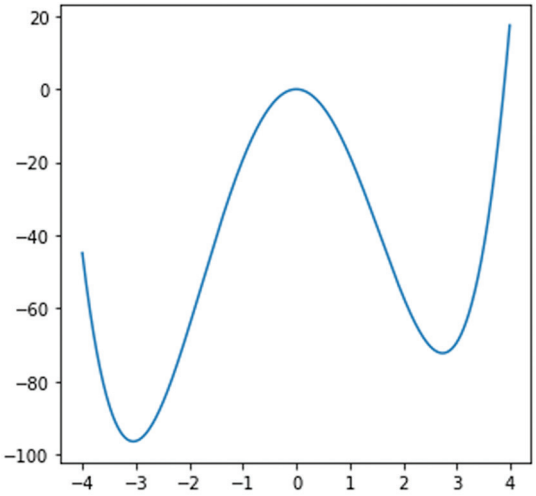


Рис. 11.15. Функция, имеющая более одного минимума (1,2x4 + 0,5x3 – 20x2)

В стартовой точке $x = -5$ вы получите:

$$\frac{dy}{dx} = 125,$$

что означает, что исходная функция в точке $x = -5$ наклонена вниз так, что функция уменьшается на 125 при изменении аргумента на 1. Если производная функции положительная, вы должны двигаться вправо, чтобы найти x, который даст минимальный результат.

Для проверки посмотрите на рис. 11.14, где видно, что при подстановке $x = 2$ в производную вы получите $dy / dx = 43$, т. е. двигаться нужно влево. Если имеется функция с несколькими переменными, как на рис. 11.13, это делается для каждой из переменных.

Как определить следующую точку

Мы определили направление, а какую теперь взять следующую точку? Как сделать следующий шаг? Возьмем, например, это уравнение:

$$x' = x - a * \frac{dy}{dx}.$$

Не переживайте из-за греческих букв: это альфа (α). В контексте градиентного спуска это называется скоростью обучения. Это означает, насколько большой шаг вы берете для перехода к следующей точке.

Опять же, здесь нет никаких правил, но, взяв слишком большой шаг, можно промахнуться мимо точки минимума. Посмотрите на шаг 3 на рис. 11.14 – если у вас будет шаг больше 5, вы пропустите минимум и будете двигаться по склонам взад-вперед. А со слишком малым шагом вы никогда не дойдете до места. Если у вас более одной переменной, это нужно делать для каждой переменной.

Когда заканчивать?

Сложно сказать, когда считать задачу выполненной (поэтому это *эвристический* метод). Тут важно, чтобы ваша целевая функция, которую вы пытаетесь оптимизировать, становилась все меньше и меньше. Например, если вы делаете шаг и функция уменьшается лишь на 0,0001, возможно, пора остановиться. Или можно задать фиксированное количество итераций (шагов) и закончить по их истечении.

11.5.3. Стохастический градиентный спуск

Градиентный алгоритм спуска, описанный ранее, также известен как *пакетный градиентный спуск*, потому что вы вычисляете значение ошибки всякий раз, когда перемещаете значение параметров, – а это довольно долго. Учтите, что набор данных, который вы используете, содержит не менее 601 263 оценок. Каждый раз, когда алгоритм градиентного спуска делает итерацию, вам нужно вычислить все 601 263 вычитаний, как показано в листинге ниже, где используется модель Django.

ЛИСТИНГ 11.10. Подсчет оценок

```
In[1]: Rating.objects.all().count()
Out[1]: 601263
```

Другой способ, который также эффективен с точки зрения производительности, называется *стохастический градиентный спуск*, где единовременно рассматривается одна оценка. Алгоритм выглядит следующим образом. Для каждой оценки *r (u,i)*:

- рассчитывается прогноз оценки $e = r_{ui} - q_i p_u$. Подробнее об этом поговорим дальше;
- обновляется значение x так, что $x = x - a \times e$, где α – скорость обучения.

Тогда вы сможете пройти по всем оценкам за одну или несколько итераций! Но тут уже много математики. Я надеюсь, что вы усвоили что-то, потому что это будет полезно в следующих примерах, где используется стохастический градиентный спуск.

Если вы заинтересуетесь глубоким обучением, прочитав мою книгу[1], то градиентный спуск будет важной частью обучения нейросетей.

11.5.4. Перейдем, наконец, к факторизации

Что делать, если попробовать проигнорировать все пустые ячейки и рассчитывать RMSE, глядя только на имеющиеся оценки, избегая тем самым проблему разреженности матрицы. И как заполнить пустые ячейки? Не огорчайтесь – все наши знания про SVD нам еще пригодятся. Доверьтесь мне.

[1] Я вот сейчас читаю «Machine Learning with TensorFlow» Нишанта Шукла (Manning, 2018).

Наша цель состоит в том, чтобы взять все ваши оценки и создать две матрицы таким образом, чтобы *i*-я строка матрицы факторов элементов умножалась на *u*-й столбец матрицы факторов пользователей, и в результате должна быть матрица, похожая на реальные оценки. Вернувшись к задаче наименьших квадратов, вы можете сказать, что нам нужно найти такие матрицы *Q* и *P*, которые минимизируют следующее уравнение для всех известных оценок. Это можно реализовать с использованием алгоритма стохастического градиентного спуска:

$$\min_{p,q} \sum_{(u,i)\in K} \left(r_{u,i} - q_i p_u\right)^2,$$

где:

- q_i – это *i*-я строка в матрице факторов элементов *Q*;
- p_u – *u*-й столбец в матрице факторов пользователей P.

Для каждой оценки вы будете обновлять матрицы Q и P, а точнее, строку и столбец в соответствующих матрицах. Прежде чем начать, вы должны решить, сколько признаков вы хотите в конечном итоге получить.

В методе Funk SVD у вас нет матрицы сигма для вычисления энергии, так что придется запускать метод на нескольких различных количествах признаков и выбрать лучший вариант. Вы можете использовать следующий алгоритм для каждого *f* в признаке, а затем продолжать до конца. (Я объясню, что делает этот алгоритм, а затем покажу, как реализовать это на Python.)

Для каждой оценки r_{ui} среди оценок нужно вычислить:

$$e_{ui} = r_{ui} - q_i p_u;$$
$$q_i \leftarrow q_i + \gamma * \left(e_{ui} * p_u - \lambda * q_i\right);$$
$$p_u \leftarrow p_u + \gamma * \left(e_{ui} * q_i - \lambda * p_u\right),$$

где:

- γ – скорость обучения;
- λ – регуляризация.

Во-первых, это уравнение рассчитывает, как сильно предсказанная оценка отличается от реальной. Вы можете использовать эту ошибку для исправления матриц *Q* и *P*. Теперь можно обновить два вектора q_i и p_u, используя рассчитанные ошибки и умножив их на скорость обучения (γ), которая часто близка по значению 0,001, и вычесть ее собственный вектор, умноженный на коэффициент регуляризации (это делается, чтобы сохранить длину (значения) малого вектора).

Теперь, когда вы один раз прошлись по всем известным оценкам, у вас будет две матрицы, которые могут быть использованы для прогнозирования оценок. Хорошо бы перетасовать список оценок, прежде чем выполнять этот алгоритм, поскольку тенденции в оценках могут привести к странностям в факторизации[1].

[1] Рандомизация для упорядочивания оценок.

11.5.5. Добавление отклонений

В предыдущем разделе мы говорили о тенденциях. Даже если уравнение уже сложновато, их стоит добавить. Что будет, если у нас будут и факторы пользователей (строка в матрице *P*), и отклонения?

Я представляю это так: пользователю нравится определенный тип фильмов, который закодирован в факторах пользователя, в то время как негативный (или позитивный) оценщик кодируется через отклонение. Прогноз оценки – это сумма четырех вещей, показанных на рис. 11.16.

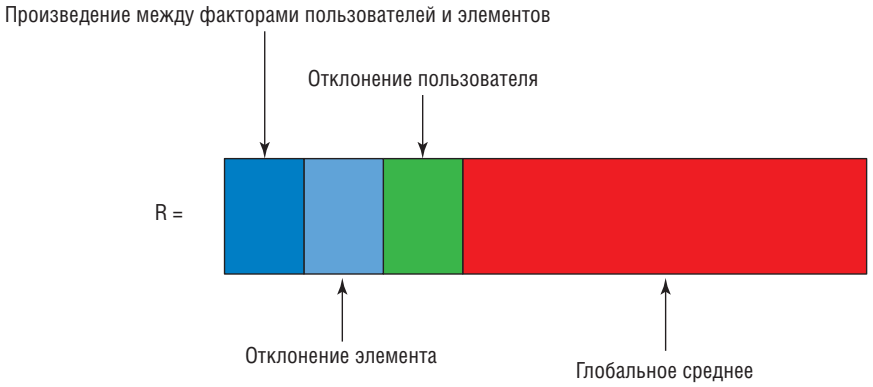


Рис. 11.16. Прогноз оценки – это комбинация четырех вещей

Если добавить это в уравнение, новая функция, которую вы хотите минимизировать, получится следующей:

$$\min_{b,p,q} \sum_{(u,i)\in K} \left(r_{iu} - \mu - b_u - b_i - q_i p_u\right)^2.$$

Теперь ваш алгоритм выглядит следующим образом (это является продолжением предыдущего). Для каждого признака f выполняется:

$$b_u \leftarrow b_u + \gamma \times \left(e_{ui} - \lambda \times b_u\right);$$

$$b_i \leftarrow b_i + \gamma \times \left(e_{ui} - \lambda \times b_i\right);$$

$$q_i \leftarrow q_i + \gamma \times \left(e_{ui} \times p_u - \lambda \times q_i\right);$$

$$p_u \leftarrow p_u + \gamma \times \left(e_{ui} \times q_i - \lambda \times p_u\right),$$

где:

γ – скорость обучения;

λ – регуляризация.

Даже если вы не собираетесь тащить слишком много через эти уравнения, стоит знать, что это все часть стохастического градиентного спуска и уравнения найдены взятием производной от квадрата ошибки.

11.5.6. Как начать и когда остановиться

Теперь, когда вся математика и структура у нас на месте, пришло время поговорить об искусстве машинного обучения, потому что одной математикой тут не обойтись: здесь также должно быть понимание того, как инициализировать параметры и какие задавать константы. Проблема любого искусства заключается в том, что ему трудно научить, потому что тут нет правильных или неправильных вещей. Подобрать хорошие параметры – целое искусство, но Симон Фанк сумел описал, какие значения ему понравились: скорость обучения знак $\gamma = 0{,}001$, регуляризация $\lambda = 0{,}02$, 40 признаков. Предположительно, это дало ему третье место в конкурсе Netflix, о котором мы говорили в начале этой главы.

Скажем пару слов о том, как определить хорошие значения для каждого из ваших параметров. Разница между тем, что мы делаем здесь, и оценкой, рассмотренной в главе 9, заключается в том, что нам нужна не только большая точность (это будет понятно после тонкой настройки параметров), но мы также сосредотачиваемся на обучении алгоритма, чтобы понять задачу. Это звучит сложно, но давайте вспомним вашу цель – предоставление рекомендаций для пользователей.

Задача состоит не в том, чтобы создать что-то, что не только запомнит ответы из обучающих данных, но и предскажет оценки тестовых данных. Дело в том, что вы хотите получить минимальную ошибку и на обучающих, и на тестовых данных, используемых при разложении на две матрицы признаков пользователей и элементов. Как описано в главе 9, вы разделяете данные так, чтобы можно было использовать обучающие данные для *обучения* алгоритма вашей задаче и *проинструктировать* тестовые данные, чтобы было понятно, когда извлеченных знаний будет достаточно. Если слишком мало обучите алгоритм, вы не получите результат, соответствующий задаче, а переобученный алгоритм будет слишком сильно ориентирован на обучающие данные. Тогда вместо понимания задачи алгоритм всего лишь освоит тестовые данные. Для манипулирования алгоритмом у вас есть следующие инструменты:

- *инициализация признаков* – нужно определить стартовую точку для градиентного спуска;
- *скорость обучения* – как быстро будем двигаться на каждом шагу;
- *регуляризация* – как сильно алгоритм регулирует ошибки? Должен ли он быть одинаковым для всех выражений, или нужно разделить его на признаки и отклонения?
- *сколько итераций* – насколько специализированным будет алгоритм для обучающих данных?

Есть много способов интерпретировать то, какой эффект вы получите, играя с этими параметрами. Я рекомендую вам стремиться понять воздействие каждого из них. Можно включить поиск по сетке, что означает, что вы переберете множество значений и посмотрите, что получится. Вы лучше справитесь со следующим рекомендатором, если уделите время, чтобы понять, какие нужны параметры.

Последнее предложение должно немного обеспокоить вас, потому вряд ли у вас будет набор данных Netflix, который был использован на Netflix Prize. Вам придется попробовать различные параметры, чтобы найти подходящие под ваш набор данных. В этом листинге показано, как можно проверить различные параметры. Вы можете посмотреть код в файле */builder/matrix_factorization_calculator.py*.

ЛИСТИНГ 11.11. Подсчет оценок

```
def meta_parameter_train(self, ratings_df):

    for k in [5, 10, 15, 20, 30, 40, 50, 75, 100]:
        self.initialize_factors(ratings_df, k)
        test_data, train_data = self.split_data(10, ratings_df)

        columns = ['user_id', 'movie_id', 'rating']
        ratings = train_data[columns].as_matrix()
        test = test_data[columns].as_matrix()

        self.MAX_ITERATIONS = 100
        iterations = 0
        index_randomized = random.sample(range(0, len(ratings)),
                                         (len(ratings) - 1))

          for factor in range(k):
            factor_iteration = 0
            last_err = 0
            iteration_err = sys.maxsize
            finished = False

            while not finished:
              train_mse = self.stocastic_gradient_descent(factor,
                                                          index_randomized,
                                                          ratings)

              iterations += 1

              finished = self.finished(factor_iteration,
                                       last_err,
                                       iteration_err)
              last_err = iteration_err
              factor_iteration += 1

              test_mse = self.calculate_mse(test, factor)
```

Обуч ется один ф ктор з р з

Иници лиз ция всех ф кторов и конст нт

Р зделение д нных н тестовые и обуч ющие

Продолж ем, пок не выполнится условие ост новки

К ждый ф ктор обуч ется 100 эпох, т. е. все оценки проходятся 100 р з

Со стох стическим гр диентным спуском в жно р ндомизиров ть д нные

З пуск стох стического гр диентного спуск н все обуч ющие оценки (подробнее позже)

Р счет средней кв др тической ошибки н тестовых д нных

На рис. 11.17 показан результат тестирования листинга 11.11 и каждый признак помещен на график.

После выполнения этого теста, который занял около 12 часов на моем MacBook с процессором Core i7 2,7 ГГц 2016 года, я пришел к выводу, что 40 признаков – это вполне неплохо. Мне также пришлось изменить число итераций. Фанк писал, что он обучал факторы за 120 итераций каждый. С этим набором данных я останавливался на 100 итерациях или когда предыдущая ошибка оказывалась меньше, чем текущая. Я думаю, здесь стоит дать ему слабину. По этой причине, если алгоритм начал обучение нового фактора, возможно, стоит дать ему поработать 10 итераций или около того, прежде чем остановить. Но я оставлю это испытание для вас.

Фанк также упоминает, что у него были разные представления о том, как вычислить ошибку прогнозирования. У вас много возможных комбинаций настроек, но лучше всего будет придумать гипотезы, а затем выполнить тестовые прогоны, как было показано ранее.

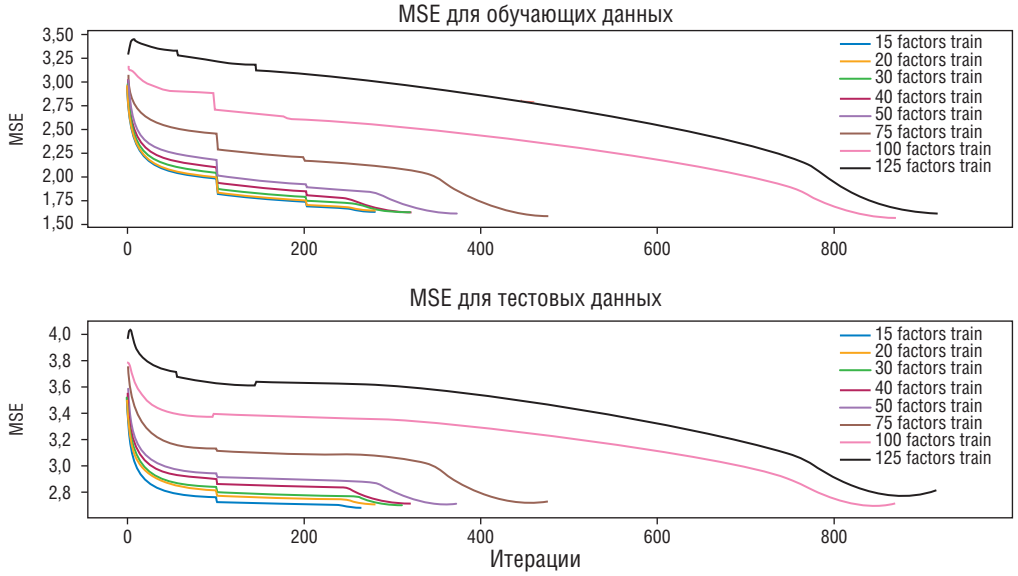


Рис. 11.17. График RMSE для каждой итерации алгоритма. Верхний график – MSE для обучающих данных, а нижний – ошибка на тестовом наборе

Другой способ взглянуть на данные – сравнить ошибку обучения с тестовой ошибкой. Это показано на рис. 11.18. Важно, чтобы линии имели угол около 45° таким образом, чтобы ошибка тестов была пропорциональна ошибке обучения.

Многие советуют запускать тест определенное число раз, например 50 или 100, в то время как другие рекомендуют смотреть на RMSE в каждой итерации и, когда изменение RMSE будет ниже определенного значения, остановиться. Если вы построите MSE, как показано на рис. 11.18, можно посмотреть на *локти* на графике. *Локоть* возникает тогда, когда вы прекращаете настройку алгоритма на известных данных и начинаете переобучение алгоритма. Линия

на рис. 11.18 показывает обучение 75 факторов. Видно, что MSE в тесте изгибается в районе 400 итераций. Здесь нужно использовать только 20 факторов, так как линии 20 факторов есть небольшой локоть в районе 275 итераций, и здесь уже можно остановиться.

Переобучение

Если ваша матрица оценок разрежена, то вы можете столкнуться с проблемой переобучения, в которой алгоритм слишком хорошо подстраивается под обучающие данные, и внезапно MSE на тестовых данных начинает увеличиваться. Например, переобучение происходит, когда матрицы U и также V вычисляют в точности правильные значения для существующих оценок, но с прогнозированием новых оценок выйдет плохо. Чтобы обойти это, можно ввести *коэффициент регуляризации*. Тогда будем минимизировать следующее:

$$\min_{uv} \sum_{(u,i)\in known} \left(r_{ui} - u_u v_i\right) + \lambda\left(\|u\|^2 + \|v\|^2\right).$$

Идея заключается в том, что вы хотите, чтобы алгоритм находил лучшие U и V, не позволяя ни одному из них становится слишком большим.

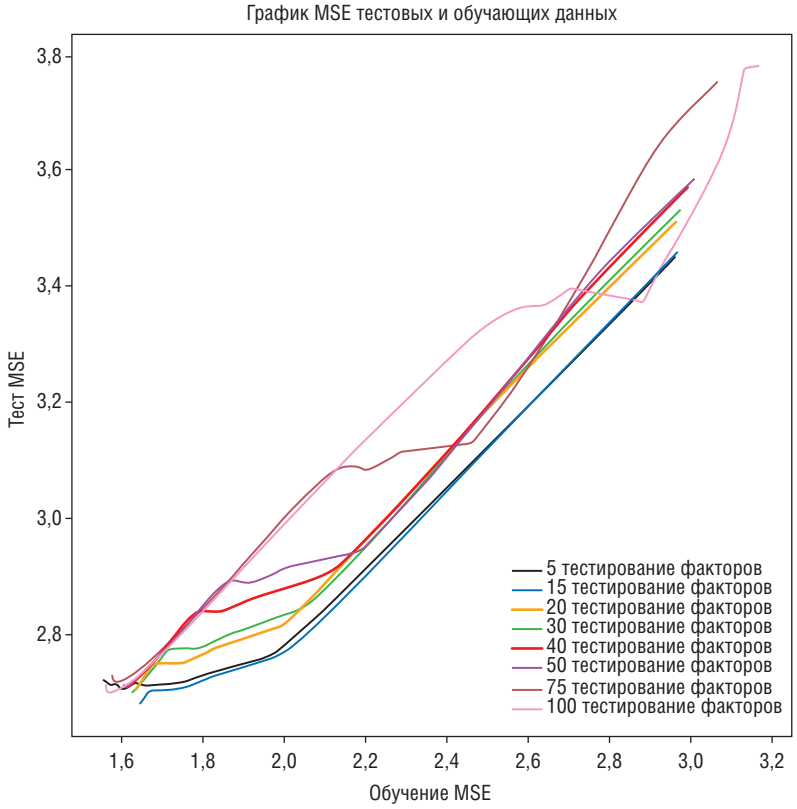


Рис. 11.18. График MSE тестовых и обучающих данных демонстрирует момент переобучения

Это делается потому, что факторизация слишком подстраивается под обучающие данные, а нам нужно, чтобы алгоритм и с новыми данными обращался нормально.

Переобучение – это интересная тема, которой требуется больше места, чем у нас тут есть[1]. В методе Funk SVD тот факт, что вы можете использовать фактор пользователя, чтобы ограничить фактор элемента, и наоборот (и то же самое с отклонениями), должен помочь справиться с переобучением.

11.6. Генерация рекомендаций с помощью Funk SVD

Когда вся эта работа выполнена, у вас будет четыре вещи:

1. Матрица факторов элементов – где каждый столбец соответствует элементу контента, описанному скрытыми факторами, которые вы рассчитали.
2. Матрица факторов пользователей – где каждый столбец соответствует пользователю, описанному скрытыми факторами.
3. Отклонение элемента – где определенные пункты считаются в целом лучше или хуже, чем другие. Отклонение – это разница между глобальным средним и средним элемента.
4. Отклонение пользователя – включает различные оценочные шкалы для разных пользователей.

С помощью этих четырех вещей вы можете рассчитать прогнозируемую оценку для любого элемента и любого пользователя, используя формулу, которую мы уже показывали:

$$\widehat{r_{u,i}} = \mu + b_u + b_i + p_i q_u.$$

И это хорошо, потому что, по сравнению с методами, которые мы изучали раньше, теперь вы можете предоставить прогнозировать оценки на все! Но рекомендации – это чуть серьезнее, чем прогноз оценки. Вам нужно определить список элементов, которые пользователь оценит высоко. У нас два способа сделать это – перебором и через окрестности.

Расчет рекомендаций «в лоб»

Тут все просто: мы рассчитываем прогнозируемую оценку для каждого пользователя для каждого элемента, затем сортируем эти оценки и выводим верхние N элементов. И пока вы тут, вы заодно можете сохранить прогнозы, чтобы они были готовы к приходу пользователя.

Это простой способ. Важно помнить, что расчет может занять некоторое время, и есть шанс, что система выполнит много вычислений, которые никогда не будут использованы. Вы могли бы оптимизировать алгоритм, но лучше было бы сохранить факторы и отклонения и использовать для расчета рекомендаций их.

[1] Подробнее в Википедии: **en.wikipedia.org/wiki/Overfitting**.

Расчет рекомендаций в окрестности

Я рассказывал об окрестностях в главе 8, поэтому повторяться не будем. Но в этой главе вместо использования данных реальных оценок мы будем использовать рассчитанные факторы. Это означает, что вы будете вычислять сходство более близких элементов и с меньшим числом измерений, что делает задачу проще.

Если вы посмотрите на пространство факторов, показанное на рис. 11.19, вы сможете создавать рекомендации по элементам или по пользователям. В любом случае мы будем использовать векторы, представляющие пользователей и элементы.

Вектор пользователя

Один из способов создания рекомендаций – это найти элементы с векторами факторов, которые близки к вектору активного пользователя. Они все находятся в одном пространстве, так почему бы нет. Но если вы сделаете это, нужно учесть, что пользователь находится посередине между всеми элементами, которые ему нравятся. Если пользователь любит только один жанр фильмов, это прекрасно. Если вы, как и я, любите и итальянские драмы, и фильмы о супергероях, то в окрестность вокруг вас может попасть вообще неизвестно что.

Элементы, которые нравятся пользователю

Возвращаемся к рис. 11.20 (квадрат из главы 8, который мы повторим здесь), пройдя через все элементы, которые пользователи оценили положительно, и найдем подобные детали или найдем аналогичных пользователей и порекомендуем элементы, которые им понравились.

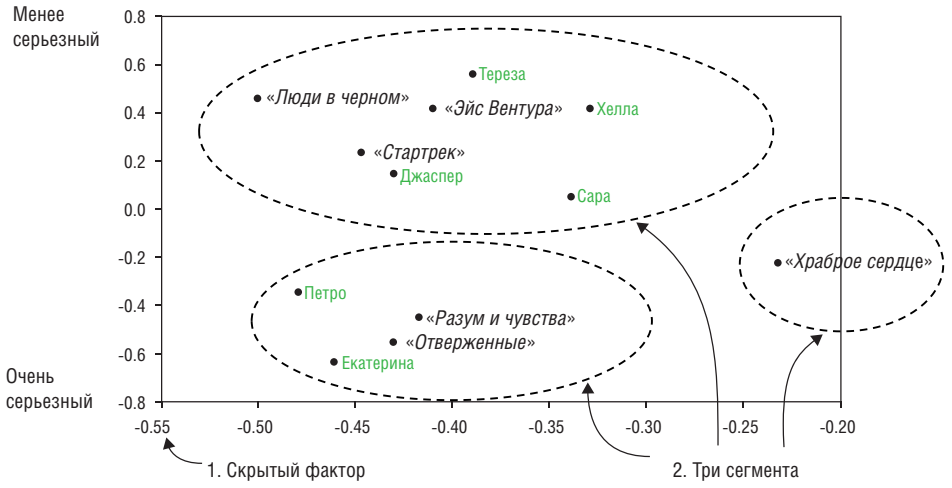


Рис. 11.19 Система координат, показывающая вариант того, как может быть интерпретировано скрытое пространство. Здесь вы можете увидеть, что Екатерине и Петро нравится «Разум и чувства» и «Отверженные». Никому не нравится «Храброе сердце»

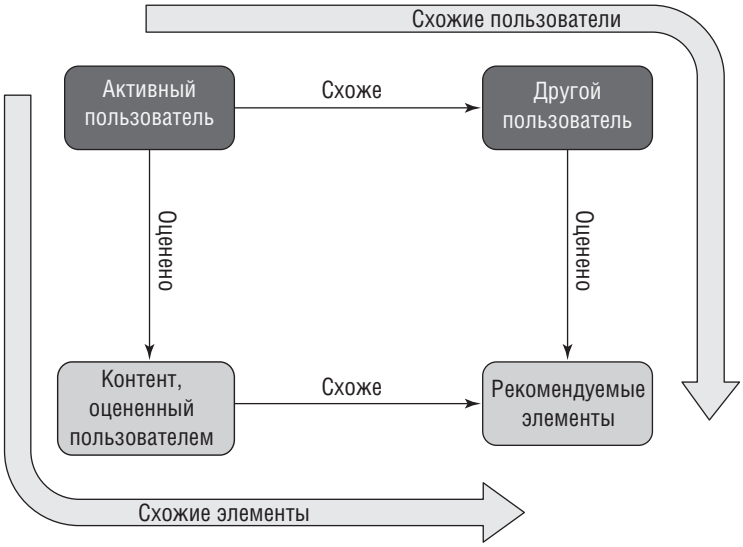


Рис. 11.20 Различные способы перейти от активного пользователя к рекомендациям – путем поиска подобных пользователей или подобных элементов

11.7. Реализация Funk SVD на MovieGEEKs

Теперь давайте вернемся к вопросу о реализации на MovieGEEKs. Мы уже рассмотрели несколько реализаций, и теперь посмотрим на рекомендации и проверим работоспособность модели.

ЛИСТИНГ 11.12. Подсчет оценок

```
Python -m builder.MatrixFactorizationCalculator
```

Вы найдете код для построения модели в файле */builder/MatrixFactorizationCalculator.py*. В следующих разделах вы увидите, что происходит внутри сборки.

Этап обучения

На этапе обучения вы инициализируете все отклонения и факторы и извлекаете оценки, как показано в следующем листинге. Поскольку мы проходим по оценкам несколько раз, лучше заранее загрузить их и держать их в памяти (если они влезают). Вы найдете этот код в */builder/matrix_factorization_calculator*.

ЛИСТИНГ 11.13. Подсчет оценок

```
def initialize_factors(self, ratings, k=25):
    self.user_ids =     set(ratings['user_id'].values)
    self.movie_ids = set(ratings['movie_id'].values)
```

Созд ние н бор ID пользов телей

Созд ние н бор ID фильмов

```
    self.u_inx = {r: i for i, r in enumerate(self.user_ids)}
    self.i_inx = {r: i for i, r in enumerate(self.movie_ids)}

    self.item_factors = np.full((len(self.i_inx), k), 0.1)
    self.user_factors = np.full((len(self.u_inx), k), 0.1)

    self.all_movies_mean = self.calculate_all_movies_mean(ratings)
    self.user_bias = defaultdict(lambda: 0)
    self.item_bias = defaultdict(lambda: 0)
```

Созд ние слов ря из `user_id`, чтобы можно было использов ть м ссивы NumPy вместо п нд. Это ускоряет процесс

Созд ние слов ря из `movie_id`, чтобы можно было использов ть м ссивы NumPy вместо п нд. Это ускоряет процесс

Созд ние двух м триц ф кторов со зн чениями 0,1

Р счет средней оценки всех фильмов

Нам нужен способ, чтобы проверить ошибки между реальными и расчетными оценками, так что вам нужен метод расчета прогноза, как показано в следующем листинге, который вы можете посмотреть в */builder/matrix_factorization_calculator.py*. Метод идентичен методу прогнозирования, который будет использоваться.

ЛИСТИНГ 11.14. Реализация метода прогнозирования оценок

```
def predict(self, user, item):
    avg = self.all_movies_mean
    pq = np.dot(self.item_factors[item],self.user_factors[user].T)
    b_ui = avg + self.user_bias[user] + self.item_bias[item]
    prediction = b_ui + pq

    if prediction > 10:
        prediction = 10
    elif prediction < 1:
        prediction = 1
    return prediction
```

Р счет лгебр ического произведения ф кторов текущего пользов теля и элемент

Суммиров ния для прогноз

Сложение отклонений

Проверяем, что зн чение прогноз лежит между 1 и 10

Теперь вы готовы к работе с алгоритмом обучения. Это уже было описано ранее и также находится в файле */builder/matrix_factorization_calculator.py*.

ЛИСТИНГ 11.15. Алгоритм обучения

```
def train(self, ratings_df, k=20):
    self.initialize_factors(ratings_df, k)
    ratings = ratings_df[['user_id', 'movie_id', 'rating']].as_matrix()

    index_randomized = random.sample(range(0, len(ratings)),
                                     (len(ratings) - 1))

    for factor in range(k):
```

Иници лиз ция всех ф кторов и конст нт, пок з нных в листинге 11.13

Форм тиров ние д нных оценок

Перемешив ние д нных, чтобы тенденции не ск зыв лись н обучении нег тивно

Проход через все *k* ф кторов

```
    iterations = 0
    last_err = 0
    iteration_err = sys.maxsize
    finished = False

    while not finished:
      start_time = datetime.now()
      iteration_err = self.stocastic_gradient_descent(factor,
                                                      index_randomized,
                                                      ratings)
      iterations += 1
      finished = self.finished(iterations,
                               last_err,
                               iteration_err)
      last_err = iteration_err
    self.save(factor, finished)
```

Сколько итераций обучения нужно?

Как объяснялось ранее, нет простого способа решить, когда останавливать обучение. В листинге ниже показан метод, который я использовал во время обучения, и вы можете увидеть этот фрагмент кода в сценарии */builder/matrix_factorization_calculator.py*.

ЛИСТИНГ 11.16. Алгоритм обучения

```
def finished(self, iterations, last_err, current_err):

    if iterations >= 100 or last_err < current_err:
        print('Finish w iterations: {}, last_err: {}, current_err {}'
              .format(iterations, last_err, current_err))
        return True
    else:
        self.iterations +=1
        return False
```

Движемся д льше: увеличив м число итер ций

Если прошло больше 30 итер ций и р зниц между ошибк ми меньше 1, можно ост новиться и бр ться з следующий ф ктор

Сохранение модели

Вы не хотите потерять всю работу, поэтому стоит сохранить модель в несколько файлов. Сохранение модели в формате JSON – один из способов сделать это в зависимости от того, как вы хотите использовать ее. Я выбрал сохранять каждую из матриц факторов и отклонения в файл. Вы можете просмотреть код в файле */builder/matrix_factorization_calculator.py*.

ЛИСТИНГ 11.17. Сохранение матриц факторов

```
def save(self):
    print("saving factors")
```

```
with open('user_factors.json', 'w') as outfile:
    json.dump(self.user_factors, outfile)
with open('item_factors.json', 'w') as outfile:
    json.dump(self.item_factors, outfile)
with open('user_bias.json', 'w') as outfile:
    json.dump(self.user_bias, outfile)
with open('item_bias.json', 'w') as outfile:
    json.dump(self.item_bias, outfile)
```

JSON в файле работает не так быстро, но это зависит от того, что вы хотите делать дальше. Если вы хотите вычислить рекомендации перебором и сохранить все предсказанные оценки в базу данных, то вам необходимо взять произведение для всех пользователей и элементов. Сохраните значения выше определенного порога в базу данных и просматривайте их, когда это необходимо.

Другой способ заключается в вычислении сходства с использованием коэффициента Отиаи, как мы делали в главе 8. Затем рассчитываем оценки так же, как мы делали в этой главе. В следующем разделе мы реализуем более медленную версию, которая загружает файлы каждый раз, когда необходимо сгенерировать рекомендации.

Онлайн-этап

В листинге 11.18 показано, как сгенерировать рекомендации путем расчета прогноза и их сортировки. Этот фрагмент кода можно найти в файле */recs/funkCP_recommender.py*.

ЛИСТИНГ 11.18. Сортировка прогнозов по убыванию: recs/funksvd_recommender.py

```
    def recommend_items_by_ratings(self, user_id, active_user_items, num=6):

    rated_movies = set(active_user_items.values('movie_id'))
    user = self.user_factors.loc[user_id]

    scores = self.item_factors.dot(user)
    scores.sort_values(inplace=True, ascending=False)
    result = scores[:num + len(rated_movies)]

    recs = {r[0]: r[1] + self.user_bias[user_id] + self.item_bias[r[0]]
        for r in zip(result.index, result) if r[0] not in rated_movies}

    sorted_items = sorted(recs.items(),
                      key=lambda item: -float(item[1]['prediction']))[:num]

    return sorted_items
```

Берутся идентификаторы всех фильмов, которые пользователь уже оценил. Покажем их еще раз не нужно

Факторы пользователей загружаются из файлов, которые вы сохранили во время обучения

Сортировка оценок по убыванию

Берется верхняя часть списка в нужном количестве без уже оцененных пользователем элементов

Создание формата данных, которые фронтенд отрезает от возвращаемого числа

Создание словаря идентификаторов контент – прогноз оценки

Выполните ручное тестирование и проверьте рекомендации для какого-нибудь пользователя, а затем посмотрите на пользователя 400005, чей профиль предпочтений показан на рис. 11.21.

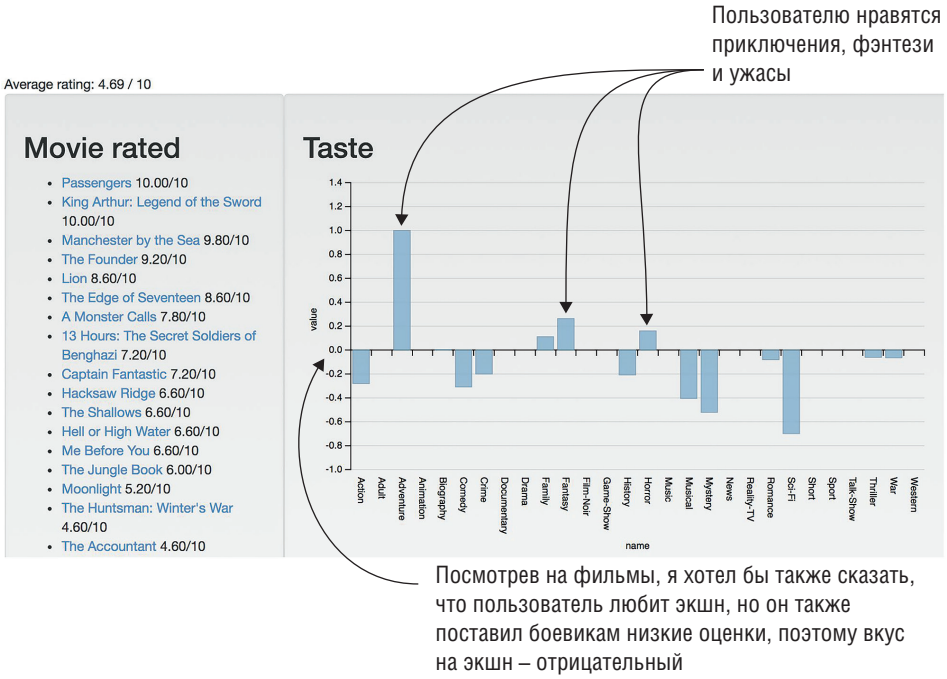


Рис. 11.21. Профиль предпочтений пользователя 400005 (http://localhost:8001/analytics/user/400005/)

Пользователь получает рекомендации, показанные на рис. 11,22 (скриншот из аналитической части MovieGEEKs: **http://localhost:8001/analytics/user/400005/**).

Проблема с поиском рекомендаций одного пользователя заключается в том, что она может оказаться одним из немногих, кому система дает хорошие рекомендации, уверенным тут быть нельзя. Именно поэтому единственный реальный способ увидеть, хороша ли модель, – это пустить ее в производство или хотя бы показать кому-то.

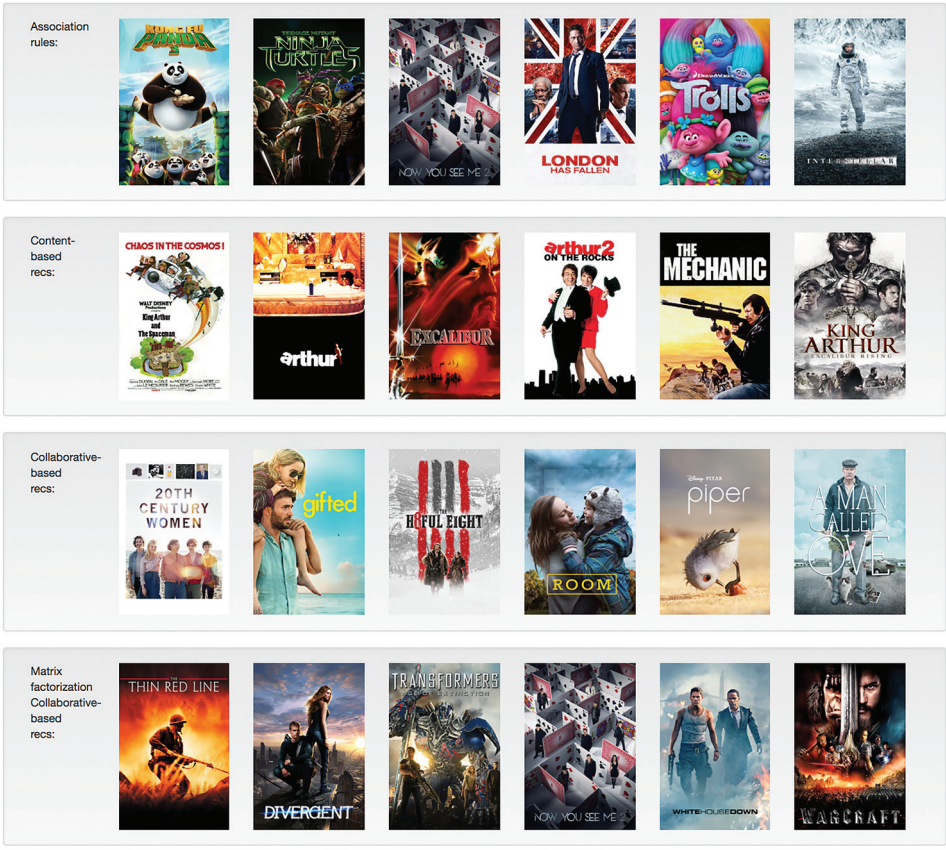


Рис. 11.22. Рассчитанные нами рекомендации: правила ассоциации на основе контента и совместной фильтрации, и, наконец, рекомендации после матричной факторизации. В основном отобрался экшн, но это вроде бы подходит тому, что пользователь смотрел (но это субъективное мнение)

11.7.1. Что делать с неподходящими рекомендациями

В одной из предыдущих итераций рекомендатора я получил что-то, что дало хороший результат при оценке, и я думал, что все было готово для запуска. Но потом я наткнулся на пользователя, который любит мультфильмы, например «Кунг-фу Панда» и «ЛЕГО-Бэтмен», но метод Funk CPA дал хорошие рекомендации, показанные на рис. 11.23. Не посмотрев эти рекомендации, я решил, они выглядят немного странно.

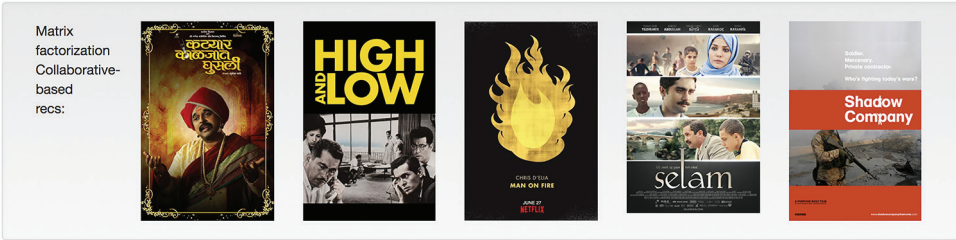


Рис. 11.23. Несколько посторонних фильмов уже проникли в рекомендации

Оказывается, что эти фильмы далеки и от «Панды», и от «Бэтмена». Вкусы пользователя указывают на приключения, анимацию, детективы, фантастику, триллеры. Далее я увидел, что у всех пяти фильмов значение отклонения было выше 5. Алгоритм берет скалярное произведение между факторами пользователя и элемента, а затем отклонения элементов прибавляются перед сортировкой. Я оказался перед дилеммой, нужно ли сортировать элементы перед прибавлением, или нет.

Если заставить алгоритм сортировать элементы отклонения элемента, то он по-прежнему будет влиять на рекомендации, но только в пределах списка элементов, которые отнесены к подобным. Теперь я получил рекомендации, показанные на рис. 11.24.

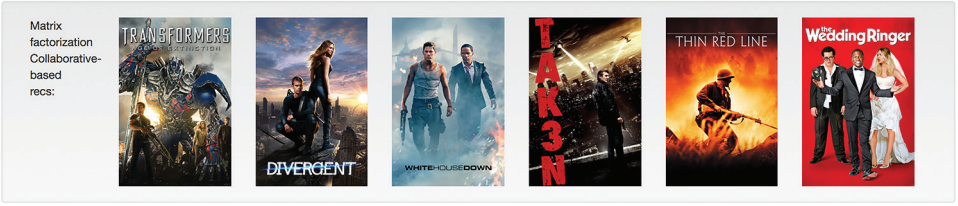


Рис. 11.24. Добавление отклонения после сортировки позволяет получить лучшие рекомендации

Еще одна вещь, с которой я экспериментировал, – я уменьшал скорость обучения для отклонений, что также помогло с этой проблемой, потому что отклонения получились меньше, и выросло разнообразие пользователей и элементов. Что тут лучше – дело вкуса, и над этим можно корпеть долго и упорно. Трудно сказать, какие настройки лучшие, так что вам стоит экспериментировать дальше.

11.7.2. Поддержание актуальности модели

Модель, которую вы создали, быстро устаревает. В идеале новый прогон нужно начинать сразу же, но, в зависимости от того, как часто изменяются ваши данные, вы можете выполнять перерасчет вашей модели раз в день или даже один раз в неделю. Но помните, что каждый раз, когда пользователь взаимодействует с новым элементом, это должно отражаться на вкусах пользова-

теля. Если выполнять расчет, также надо не забыть удалить из рекомендаций элементы, оцененные пользователем. Наконец, помните, что если вы использовали неявные данные, то наличие оценки не означает, что пользователь просмотрел фильм.

11.7.3. Более быстрая реализация

Матричная факторизация выполнялась с использованием градиентного спуска, но, если у вас есть миллионы элементов и пользователей, процесс может быть медленным. Другим способом является метод чередующихся наименьших квадратов (ALS), который не так точен, как градиентный спуск, но он должен работать достаточно хорошо в любом случае.

11.8. Явные данные против неявных данных

Если у вас есть набор неявных данных, вы могли бы сделать, как и в главе 4, и вывести оценки из них. Кроме того, вы можете использовать эти данные непосредственно, слегка изменив прежний алгоритм. Для получения более подробной информации о том, как сделать это, я рекомендую вам прочитать книгу «Совместная фильтрация для обратной связи неявных данных» Ифаня Ху и др. (**yifanhu.net/PUB/cf.pdf**)

Если вы хотите перейти непосредственно к реализации, посмотрите на неявную структуру, которая является быстрой для неявных данных и совместной фильтрации[1]. Я слышал, что он используется в некоторых производственных средах, где у вас много данных. (Я могу сказать вам, где, но тогда мне придется вас убить.)

Как уже упоминалось, в этой реализации используется несколько иных алгоритмов, отличных от описанных здесь. Отличным упражнением может быть просмотр этого кода, чтобы понять альтернативные способы сделать это.

11.9. Оценка

Когда вы работаете с Funk SVD или другим алгоритмом машинного обучения, у вас часто много параметров, которые надо настроить перед началом фактической оценки. Ранее мы уже говорили об ошибке обучающих данных и сравнивали ее с ошибкой на тестовых данных. Затем, когда вы закончите настройку этих переменных, вы можете оценить алгоритм. Многие сказали бы, что это одно и то же. Но потом все равно придется проверить алгоритм на новых данных, чтобы увидеть, как это работает.

Вы не хотите искать гиперпараметры и выполнять кросс-проверку, потому что это занимает слишком много времени. Вы почти всегда хотите использовать кросс-проверку для оценки эффективности алгоритма. Принцип точно такой же, как описано в главе 9 и показано на рис. 11.25.

[1] Подробнее: **github.com/benfred/implicit**.

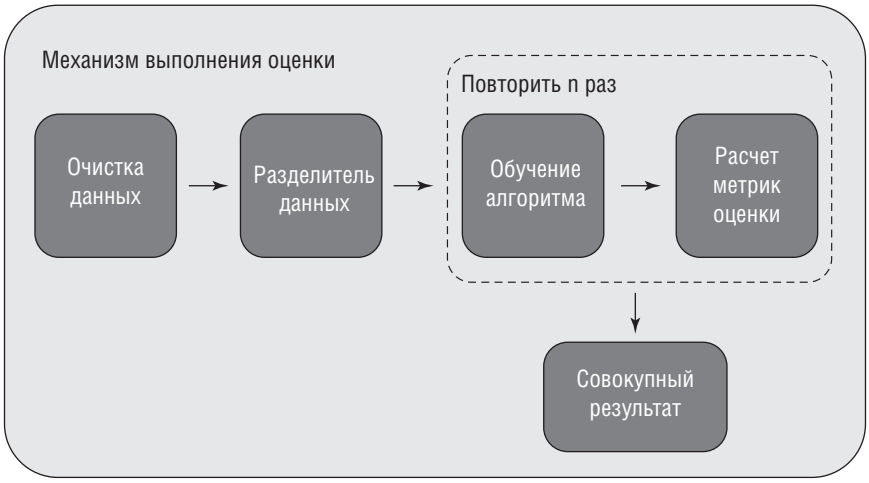


Рис. 11.25. Схема выполнения оценки

Нужно очистить данные, чтобы иметь только релевантную информацию, затем нужно разделить пользователей на *k* частей. Тогда для каждой из этих частей вы разделяете данные на обучающие и тестовые. Обучающие данные части объединяются и добавляются к *k* – 1 данных из других частей и используются для обучения алгоритма. Тогда алгоритм тестируется с использованием одной из метрик, описанных в главе 8, и, наконец, вы агрегируете результат. Вы можете запустить оценку, выполнив команду:

ЛИСТИНГ 11.19. Оценка

```
> python -m evaluator.evaluation_runner -funk
```

Эта команда создает файл CSV с данными, используемыми для отображения оценок. Рекомендую проверить код и посмотреть, как это делается.

Этот сценарий не запускает кросс-проверку из коробки, вам нужно настроить код для его запуска. По умолчанию задано 0 частей, что означает отсутствие перекрестной проверки. Я провел оценку на модели с 40 факторами, расчитал точность и вспомнил, что вы сделали в двух предыдущих главах. Результат показан на рис. 11.26.

Глядя на результат факторизации, я думаю, здорово видеть, что она работает на 100 % пользователей, но в рекомендации попадает лишь 0,11 % элементов, т. е. чуть более 3200 элементов. (Я немного подправил скорость обучения отклонения. При значении 0,0005 модель охватывала лишь около 200 элементов, а при 0,002 – более чем 3200 элементов. Но до полного набора 28700 элементов еще далеко.) Одной из причин таких низких процентов может быть необходимость добавить больше факторов. Но опять же, этим можно поиграть, и это отличное упражнение для вас.

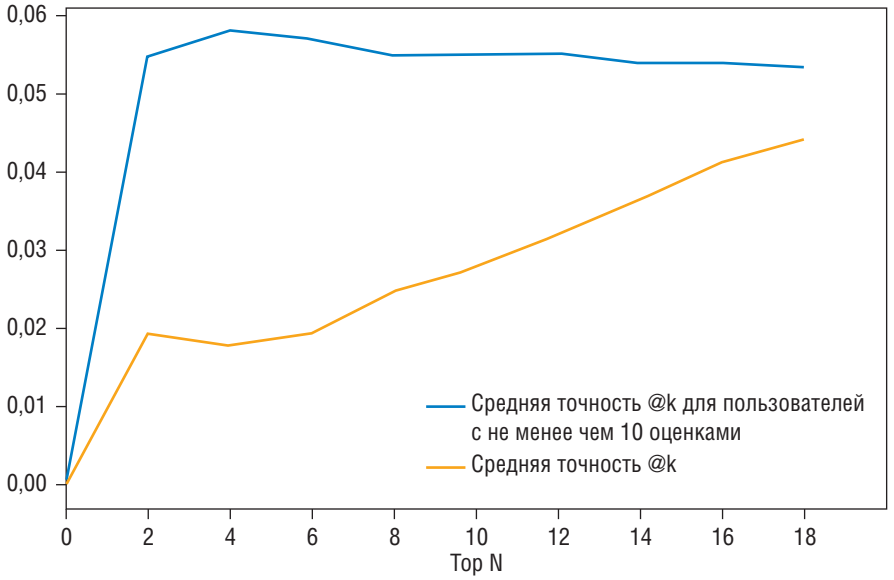


Рис. 11.26. Средняя точность метода Funk SVD. Нижняя линия – тест всех пользователей, а верхняя – только пользователей, которые поставили не менее 10 оценок

Точность на рис. 11.26 намного лучше, чем алгоритм на основе контента, где и запоминаемость, и точность были наполовину хуже. Дело в том, что вас мог сбить с толку тот факт, что модель с окрестностями имеет лучшую запоминаемость. Но разница в том, что у вас охват 100 % пользователей, т. е. пользователи, которые оценили только один элемент, также включены. А из модели для расчета окрестностей мы их удалили. Если удалить всех пользователей с несколькими оценками, как мы делали в главе 8, то вы, вероятно, получите высокую точность, как показано на рис. 11.27.

11.10. Эксперименты с моделью Funk SVD

Есть много разных крутилок и свистелок, которые можно поменять и отладить при использовании алгоритма Funk SVD. Сложность тут в том, что они все зависят друг от друга, т. е. оптимизация одного параметра не всегда окажется оптимальной, если вы измените другой фактор. Вы должны попробовать все сочетания, чтобы быть полностью уверенными, что нашли лучшее. Тут можно воспользоваться поиском по сетке или чем-то аналогичным[1].

Добавление оценок в обучающие данные для пользователей, которые поставили только одну оценку, не сильно поможет алгоритму совместной фильтрации, поэтому было бы хорошо убрать их. Их вклад остается только в средней оценке фильмов.

[1] Подробнее см. **scikit-learn.org/stable/modules/grid_search.html**.

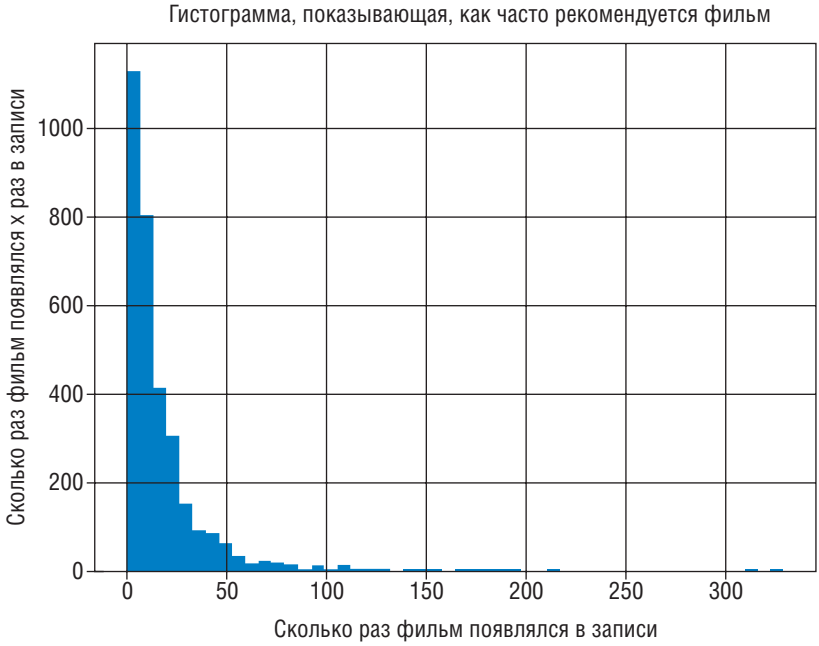


Рис. 11.27. График показывает, сколько фильмов было оценено определенное количество раз. Исходя из этого, вы можете увидеть, что пользователю рекомендуется всего около 1000 фильмов

Факторы элементов и пользователей отображают пользователей и элементы с использованием скрытых факторов, на которых выполнялось обучение. Хорошо бы проверить, что эти факторы не слишком велики. Поводом для волнения было бы, если бы в каком-то векторе было значение, далекое от 1. Вы можете настроить это с помощью регуляризации – чем больше регуляризация, тем больше факторов будет иметь значение, близкое к 0.

Инициализация отклонений также имеет значение, поскольку она определяет, насколько велика будет ошибка прогнозирования факторов в самом начале. Не нужно, чтобы она была слишком велика. Учтите, что, если у вас есть средняя оценка всех фильмов, добавление отклонений пользователя и элемента не должно исключать среднее за пределы оценочной шкалы. В вашем наборе данных используется оценочная шкала от 1 до 10. Средняя оценка равна около 7, поэтому отклонения элементов и пользователей не должны превышать 1,5.

Определение правильного количества итераций – это тоже дело вкуса. Если вы выполняете слишком много итераций на ранних факторах, вы рискуете, что все сигналы будут вытесняться в первое измерение, а остальные факторы будут иметь слабый сигнал. При слишком малом числе итераций векторы никогда не будут выровнены правильно. В зависимости от того, сколько итераций вы ставите, вам нужно также выбрать оптмальную скорость обучения – чтобы не промахнуться мимо оптимальной точки.

Это короткий список вещей, с которыми можно побаловаться. Но их намного больше. Не забудьте выбрать хорошую метрику качества и пользуйтесь ей. Низкое значение RMSE/MSE не обязательно даст лучшие рекомендации, но зато показывает, как прийти к этому. Опять же, не забудьте выдвинуть гипотезу и убедитесь, что вы знаете, как оценить результат, и сделайте процесс тестирования как можно проще.

Это был сложный раздел, и вы многому научились. Матричня факторизация – это сложная тема сама по себе, мы коснулись ее лишь поверхностно. Но если вы поняли эту главу, то с новыми знаниями сможете сделать гораздо более сложные вещи.

Резюме

- SVD – способ выполнить матричную факторизацию. Этот метод хорошо выполняется во многих библиотеках. Его недостатком является то, что он не будет работать, если матрица не заполнена, т. е. в ней не должно быть пустых ячеек. Вы должны будете чем-то заполнить эти ячейки, а это довольно трудная задача, ведь неизвестно, что там должно быть.
- Базисные предикторы – это способ заполнить эти пустые ячейки. То же самое верно и для элементов.
- Базисные предикторы используются для получения отклонений пользователей и элементов.
- Саймон Фанк разработал метод Funk SVD, который позволяет использовать разреженную матрицу оценок.
- Градиентный спуск и стохастический градиентный спуск – это крутые инструменты для решения задачи оптимизации, например для подготовки и оптимизации Funk SVD.

Глава 12

С каждого по способностям – реализуем гибридный алгоритм рекомендательной системы

В данной главе мы рассмотрим самые разные вещи:

- научимся комбинировать рекомендательные алгоритмы, используя сильные и слабые стороны различных типов рекомендательных систем;
- пройдемся по общим классам гибридных рекомендаторов;
- познакомимся с ансамблями рекомендаторов;
- познакомившись с ансамблями рекомендаторов, увидим, как реализовывается особый алгоритм под названием признако-взвешенное линейное сочетание (FWLS – Feature-Weighed Linear Stacking).

Предположительно, один из самых энергоэффективных автомобилей – гибридный Toyota Prius. Эта модель Toyota сочетает в себе две известные технологии – двигатель внутреннего сгорания и электродвигатель[1]. Гибридные рекомендаторы – это то же самое. Мы объединяем разные рекомендательные алгоритмы, чтобы получить на выходе более мощный инструмент. Это позволяет не только улучшить средний результат, но и попытаться сгладить слабые места этих алгоритмов.

На рис. 12.1 показаны четыре наиболее известных класса рекомендательных систем и их источников данных. Мы говорили о каждой рекомендательной системе как об одиночном черном ящике, но это далеко не так. Чтобы генерировать рекомендации, вам нужно сочетание нескольких систем или гибрид.

Кроме того, если у вас имеется несколько источников данных, показанных на рисунке, грех не использовать их все!

[1] Подробнее: **en.wikipedia.org/wiki/Toyota_Prius**.

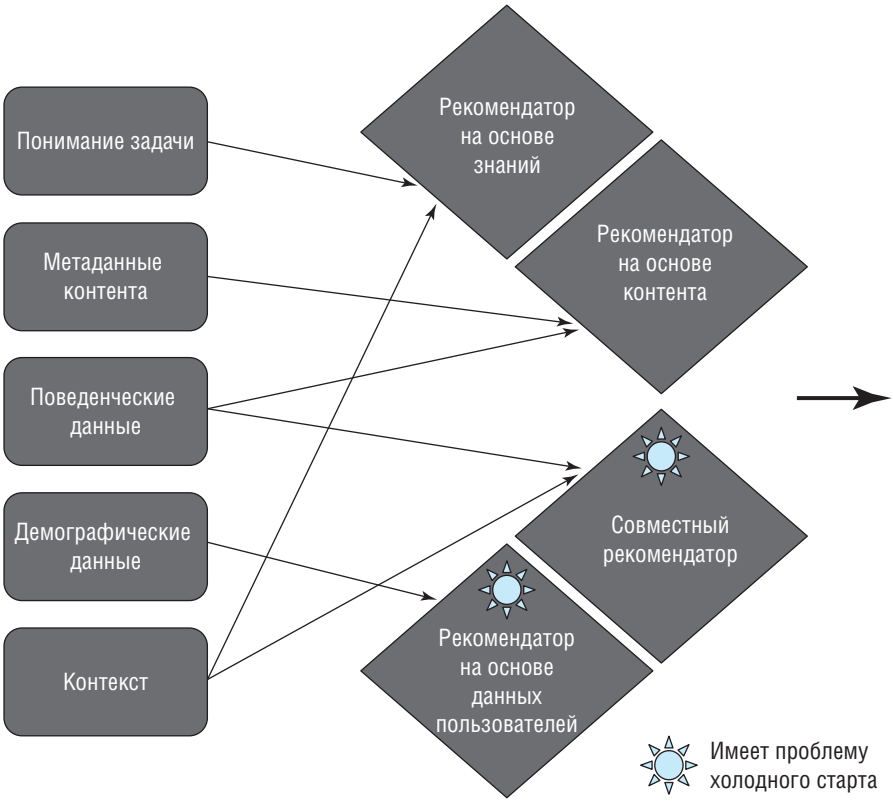


Рис. 12.1. Четыре основных класса рекомендаторов и типы данных, которые они берут на вход

Нет никаких ограничений в том, как именно комбинировать алгоритмы, и, если мы рассмотрим в этой главе все варианты, она распухнет до размера отдельной книги. Чтобы избежать этого, я начну с небольшого обзора различных категорий, которые рассматриваются чаще всего. Затем поподробнее поговорим об алгоритме FWLS. В конце мы посмотрим, как реализовать этот алгоритм на MovieGEEKs.

12.1. Сложности мира гибридов

Гибридными рекоменадаторами обобщенно называются типы рекомендаторов, показанные на рис. 12.2. Я представляю гибриды примерно так.

Монолитные рекомендаторы берут компоненты различных рекомендаторов и по-новому склеивают их вместе. Ансамбль – это нескролько работающих рекомендаторов, результаты работы которых комбинируются в одну рекомендацию. Смешанный рекомендатор возвращает результат работы сразу нескольких рекомендаторов. О каждом из этих типов мы подробно поговорим в следующих разделах.

Рис. 12.2. Три общих гибридных рекомендаторов

12.2. Монолитные рекомендаторы

Само слово «монолит» мало связано с рекомендаторами. Вместо этого оно означает нечто сделанное из цельного куска камня, как головы на острове Пасхи, или же организацию, характеризующуюся мощью, жесткостью и неповоротливостью. Короче, в описание компании это слово выглядит неприглядно, но именно так называются рекомендаторы.

В нашем случае *монолитный гибридный рекомендатор* – это Франкенштейн в мире рекомендаторов. Он состоит из частей различных типов рекомендательных алгоритмов. Обычно и сам рекомендатор состоит из множества различных компонентов, например алгоритм расчета сходства, отбор кандидатов и т. д. Монолитный рекомендатор берет компоненты из разных рекомендаторов или даже добавляет что-то свое для повышения общей производительности. На рис. 12.3 показан монолитный рекомендатор, в котором взят алгоритм сходства элементов из рекомендатора 1 и отбор кандидатов с прогнозированием из рекомендатора 2.

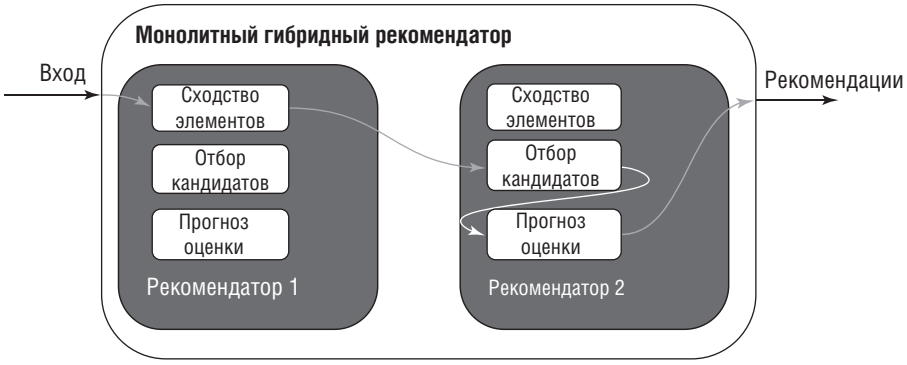


Рис. 12.3. Монолитный гибридный рекомендатор состоит из частей разных рекомендательных систем

Пример: мы можем использовать подход на основе контента, в котором находятся все подобные элементы, смешать все это с совместной фильтрацией для расчета прогноза оценок, как будто мы использовали простую совмест-

ную фильтрацию (когда мы делали персонализированные рекомендации). Возможности безграничны. В монолитный рекомендатор можно также добавить еще один шаг, как описано в следующем разделе.

12.2.1. Смешивание функций контента с поведенческими данными для улучшения алгоритмов на основе совместной фильтрации

Весь смысл гибридов заключается в том, чтобы взять все лучшее от разных типов данных. Примером монолитного гибридного рекомендатора может быть рекомендатор, основанный на совместной фильтрации, с одной дополнительной стадией предварительной обработки, где в матрицу будут добавлены оценки для дополнительных связей. Скажем, у вас есть список фантастических фильмов, показанный в табл. 12.1.

Таблица 12.1. Пример матрицы оценок

	Фильм 1	Фильм 2	Фильм 3	Фильм 4
Пользователь 1	4	4		
Пользователь 2	5	4		
Пользователь 3			2	4

Используя совместную фильтрацию в окрестности на матрице оценок, показанной в табл. 12.1, вы не получите никакого сходства между первыми двумя фильмами и двумя последними, поэтому у вас не получится ничего никому рекомендовать, так как нет нужных связей и их нужно добавить. Мы знаем, что все фильмы у нас жанра фантастики, значит, было бы хорошо добавить в матрицу другого пользователя, который любит все фильмы этого жанра. Затем можно обновить матрицу оценок, как показано в табл. 12.2.

Таблица 12.2. Пример матрицы оценок, куда добавили любителя фантастики

	Фильм 1	Фильм 2	Фильм 3	Фильм 4
Пользователь 1	4	4		
Пользователь 2	5	4		
Пользователь 3			2	4
Любитель фантастики	5	5	5	5

Это позволяет связать фантастические фильмы и рекомендовать фильмы того же жанра. Кроме того, можно использовать модель LDA, реализованную в главе 10, в которой мы создали псевдопользователя для каждой из скрытых тем, а затем добавили новые связи в наш коктейль. Чтобы понять это глубже, придется углубляться в технические детали. Для этого я рекомендую вам прочитать статью «Помощь контента в совместной фильтрации для улучшения рекомендаций» Према Мелвилле и др.[1]

[1] Подробнее: **www.cs.utexas.edu/~ai-lab/pubs/cbcf-aaai-02.pdf**.

Реализация предыдущего примера требует немалой работы, и для превращения обычного рекомендатора в монолитный в целом потребуется много усилий. Если у вас уже есть готовые рекомендаторы, вы можете попробовать смешанные гибриды или ансамбли.

12.3. Смешанный гибридный рекомендатор

Смешанный рекомендатор возвращает комбинацию результатов всех входящих в него рекомендаторов. Смешанный гибрид используется так, как я показывал в иерархии рекомендаторов.

Рассмотрим рекомендаторы с точки зрения персонализации. Сделайте первый рекомендатор максимально персонализированным, а затем продолжайте, пока не дойдете до использования популярности элементов для выдачи рекомендаций. Часто наиболее персонализированный рекомендатор дает только одну или две рекомендации, следующий рекомендатор даст несколько больше и таким образом у вас всегда будет хорошее количество рекомендаций, и при этом не в ущерб качеству. На рис. 12.4 показан смешанный гибридный рекомендатор.

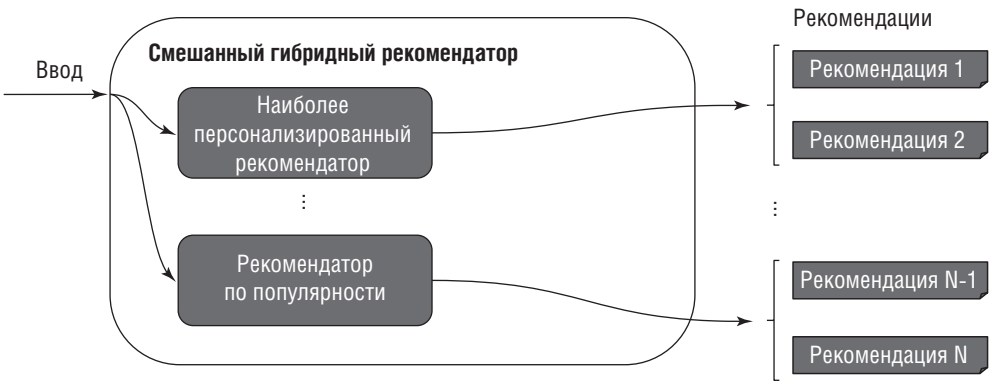


Рис. 12.4. Смешанный гибридный рекомендатор, который просто складывает выходы нескольких рекомендаторов, начиная с самого персонализированного, затем от менее персонализированного и т. д.

Если у вас есть несколько хороших рекомендаторов, каждый из которых возвращает очки в результате, то вы можете вернуть соответствующим образом упорядоченный список. Помните, что очки тоже нужно нормировать, чтобы все результаты имели один масштаб. Не отходя от идеи иметь несколько рекомендаторов сразу, давайте теперь перейдем к ансамблям.

12.4. Ансамбль

Ансамбль – это группа вещей, рассматриваемых как единое целое, а не по отдельности. То же самое для ансамбля рекомендаторов: мы берем результаты работы нескольких рекомендаторов и объединяем их.

ПРИМЕЧАНИЕ. Разница между ансамблем и смешанным рекомендатором заключается в том, что ансамбль может и не показать результат одного рекомендатора, в то время как смешанный гибрид всегда показывает все результаты.

На Ежегодной конференции по рекомендательным системам RecSys 2016 говорили о том, что если у вас есть стартап и вы хотите погрузиться в рекомендательные системы, лучше начать с матричной факторизации (тема главы 11) и добавлять рекомендаторы, создавая таким образом ансамбль рекомендаторов (см. рис. 12.5). Именно так говорил Завьер Аматриан в своем выступлении на тему «Уроки, извлеченные из создания рекомендательных систем». Почитайте, а заодно и презентацию Завьера посмотрите, там интересно[1].

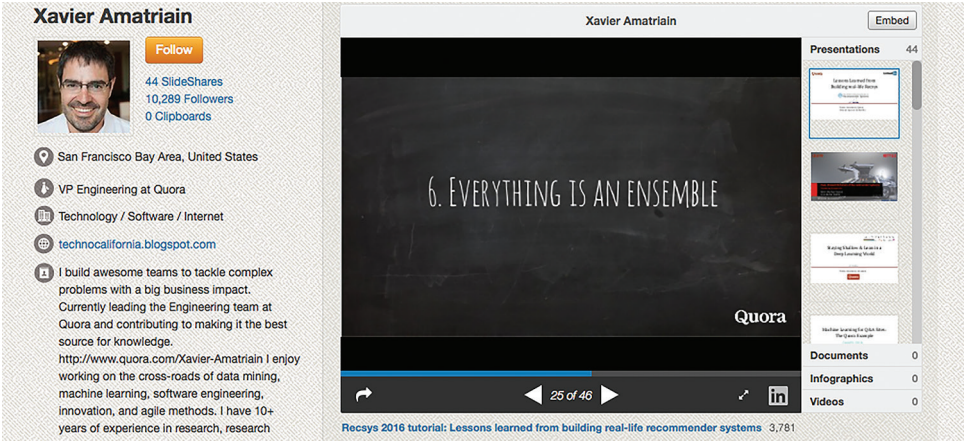


Рис. 12.5. На RecSys2016 отметили, что все – ансамбль

Если у вас уже работает два рекомендатора, скажем, на основе контента и с совместной фильтрацией, то почему бы не запустить их одновременно, как показано на рис. 12.6, а затем путем сочетания получить еще лучший результат?

Идея ансамблей заключается в использовании нескольких полноценных рекомендаторов, чьи результаты каким-то образом объединяются. Вы можете взять результат нескольких рекомендаторов и превратить их в один многими способами. Например, вы можете прибегнуть к голосованию и отсортировать элементы по его результатам.

[1] Подробнее: **www.slideshare.net/xamat**.

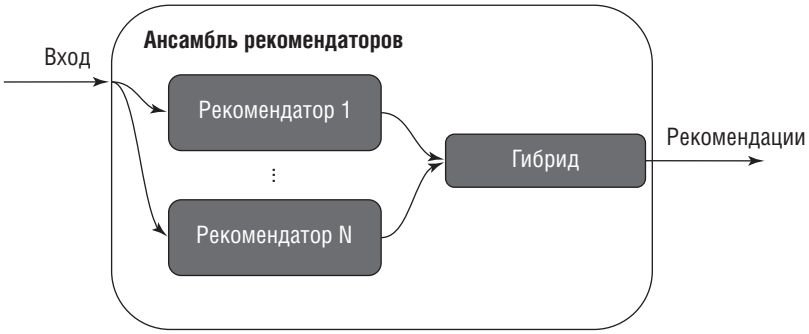


Рис. 12.6. Ансамбль – это ряд рекомендатров, чьи результаты объединяются в одну рекомендацию

На рис. 12.7 показан пример того, как может работать ансамбль. Рекомендатор 1 возвращает рекомендацию для топ-3 фильмов 1, 5 и 6. Рекомендатор 2 возвращает 5, 6 и 3. Затем гибрид возвращает 5, 6 и 1 в зависимости от того, как вы рассчитаете приоритет. Фильмы 5 и 6 точно попадут в выдачу, так как оба рекомендатора отметили их, но 6 будет чуть ниже. Затем будет рекомендован фильм 1, потому что у рекомендатора 1 он на первом месте.

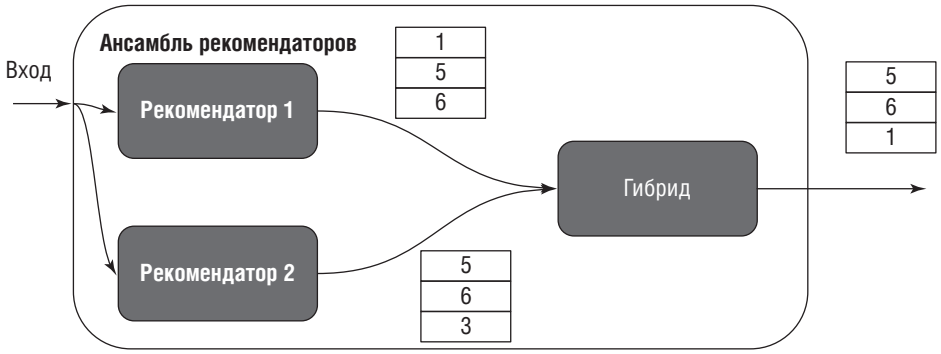


Рис. 12.7. Пример того, как может работать ансамбль

Часто применяются так называемые взвешенные ансамбли. Перейдем к ним.

12.4.1. Переключаемый ансамбль рекомендаторов

Переключаемый ансамбль рекомендатора – это когда мы выбираем лучший инструмент для работы. Если у вас есть два или более рекомендаторов, переключающийся ансамбль будет выбирать, какой из них использовать, с учетом контекста запроса.

Например, у вас может быть два разных рекомендатора для двух разных стран. Когда заходит пользователь из одной страны, выводятся результаты одного рекомендатора, и, если заходит кто-то другой страны, используется второй рекомендатор. Можно разделять и по времени суток. Может быть, один работает по утрам, а второй – вечером. Или, например, страница националь-

ных новостей в газете заполненяется последними новостями, а на культурную страницу выводятся контентные рекомендации для конкретных книг.

Вы можете переключаться между рекомендаторами в зависимости от раздела сайта, где находится пользователь. Или в самом простом виде у вас может быть два рекомендатора для пользователей, поставивших более 20 или менее 20 оценок, как показано на рис. 12.8. Пользователи с более чем 20 оценками получают рекомендации по совместной фильтрации, а пользователи с меньшим количеством оценок получают контентные рекомендации (см. главу 6).

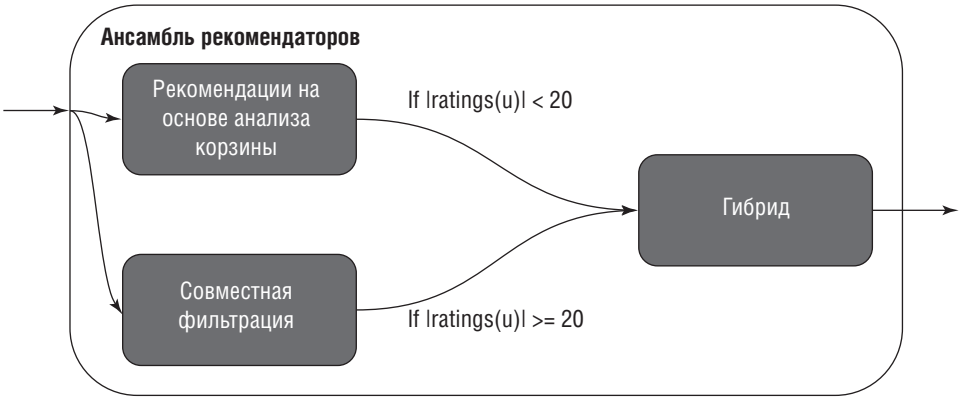


Рис. 12.8. Пример переключаемого ансамбля, где пользователи с более чем 20 оценками получают результаты от одного рекомендатора, а пользователи с менее чем 20 оценками – от другого

Если пользователь зашел на сайт под своей учетной записью, вы знаете о нем больше. Это может быть хорошей причиной использовать другой рекомендатор, не тот, что для гостей. Но если вместо этого вы хотите объединить силы различных рекомендаторов, читайте дальше о взвешенных ансамблях.

12.4.2. Взвешенный ансамбль рекомендаторов

Вспомним два алгоритма, упомянутые ранее в этой книге: совместная фильтрация (неважно, в окрестности или SVD) и фильтрация на основе контента. Фильтрация на основе контента хороша при обнаружении подобного контента. Если вы знаете, что пользователю нравится определенный жанр, вы можете использовать фильтрацию на основе контента, чтобы найти подобные элементы. Проблема заключается в том, что фильтрация на основе контента не различает хороший и плохой контент, а ориентируется только на тему и ключевые слова. Совместная фильтрация, с другой стороны, не придает никакого значения теме элемента, а смотрит только на качество и оценки других пользователей.

Вы можете использовать оба метода и попытаться объединить их сильные стороны. Но наделять их равной значимостью не обязательно – и вот тут мы подошли к идее *взвешенного гибридного рекомендатора*. Идея проста – мы обучим два различных рекомендатора и попросим их обоих сгенерировать

рекомендации. Когда два или более рекомендаторов объединяются таким образом, мы будем называть их *функциональными рекомендаторами*.

Вы можете дать всех кандидатов на вход обоим рекомендаторам, а затем взять эмпирическое среднее из двух, как показано на рис. 12.9. Для того чтобы вычислить эмпирическое среднее, вы выбираете вес, например 60/40, и тогда прогнозы этого гибрида будут рассчитываться по следующей формуле:

$$\hat{r} = 0,6 \times \hat{r}_{совместный} + 0,4 \times \hat{r}_{контент}.$$

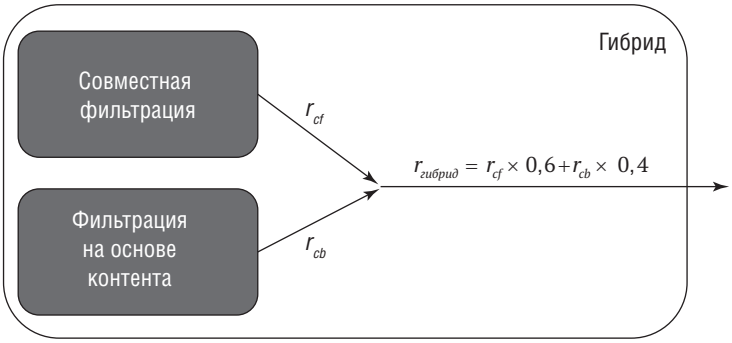


Рис. 12.9. Функционально-взвешенный гибрид, который сочетает в себе результаты совместной фильтрации и фильтрации на основе контента с весами 0,6 и 0,4 соответственно

Весовые функции можно найти несколькими способами. Самый простой способ – это угадать, но это не слишком научно и может дать странный результат. Другой способ – выполнить линейную регрессию. Вы также могли бы выполнять непрерывную корректировку весов, используя различные значения для различных групп пользователей, и посмотреть, в каком случае будет больше кликов, используя A/B-тестирование или несколько вооруженных бандитов (как упоминалось в главе 9). Давайте кратко рассмотрим линейную регрессию.

12.4.3. Линейная регрессия

Линейная регрессия – это создание функции, которая минимизирует ошибку между выходным и фактическим значением. Если у вас есть выход двух рекомендаторов и оценка от пользователя (например, данные, аналогичные табл. 12.3), то вы можете использовать линейную регрессию, чтобы найти, как сильно гибридный рекомендатор должен взвешивать каждый свой выход.

Таблица 12.3. Пример прогноза двух рекомендаторов и оценки пользователя

Элемент	RecSys1	RecSys2	Рейтинг пользователей
I1	4	6	5
I2	2	4	3
I3	5	5	5

Теперь создадим функцию *f*, которая при известном выходе двух рекомендаторов лучше предсказывает оценки пользователя. Вам нужна функция, в которой вы можете свести к минимуму разницу выходом и оценкой пользователя, например:

$$RSS = \sum_{\text{все рейтинги в обучающих данных}} \left(f(u,i) - r\right)^2.$$

Эта функция может выглядеть чуть ближе к предыдущей:

$$f(u,i) = w_1 \times RecSys1(u,i) + w_2 \times RecSys2(u,i).$$

Нужно найти такие w_1 и w_2, чтобы сумма в предыдущем уравнении была минимальной. Есть много способов определить эти значения, и некоторые из них сложные. Я думаю, мы поступим как во «Введении в статистическое обучение» (Спрингер, 2017) и двинемся дальше, отметив, что оценки коэффициентов множественной регрессии (предыдущее уравнение) имеют несколько сложную форму[1]. Надо сказать, что существует много программных пакетов, которые легко решат эту задачу[2].

Хорошо иметь вес для каждого рекомендатора, но еще лучше, чтобы эти веса изменялись в зависимости от пользователя или элемента. Этим мы и займемся в следующем разделе.

12.5. Признако-взвешенное линейное сочетание (FWLS)

Мы рассмотрели функционально-взвешенные гибриды, где комбинируется несколько рекомендаторов со статичными значениями весов. Это похоже на линейную комбинацию различных рекомендаторов, например:

$$b(u,i) = w_1 + r_{cf}(u,i) + w_2 + r_{cb}(u,i),$$

где:

- $b(u,i)$ – это прогнозируемая оценка элемента *i* от пользователя *u*;
- w_1, w_2 – веса;
- $r_{cf}(u,i)$ – прогноз совместной фильтраци;
- $r_{cb}(u,i)$ – прогноз на основе контента.

Результат можно назвать *смесью*, что также обозначает сами признако-взвешенные гибриды[3]. В этом примере мы взвесили входы двух рекомендаторов (с весами w_1 и w_2). Вы можете использовать и более двух – тут ограничений нет. В этом разделе мы рассмотрим, как превратить веса в функции. Возьмем решение из одноименной статьи Джозефа Силла и соавт.[4]

---

1 Подробнее: **www-bcf.usc.edu/~gareth/ISL/**, отличная книга по машинному обучению.

2 Например, Scikit-learn на **mng.bz/P375**.

3 Прогнозируемая оценка также иногда называется функцией.

4 Подробнее: **arxiv.org/pdf/0911.0460.pdf**.

12.5.1. Представляем веса в виде функций

Допустим, вы хотите, чтобы ваша рекомендательная система стала еще более гибкой. Например, вы хотите использовать рекомендации на основе контента, если пользователь оценил мало элементов, и совместную фильтрацию, если пользователь оценил много элементов. Чтобы сделать это, вы можете расширить предыдущий пример, заменив веса функциями, как показано на рис. 12.10.

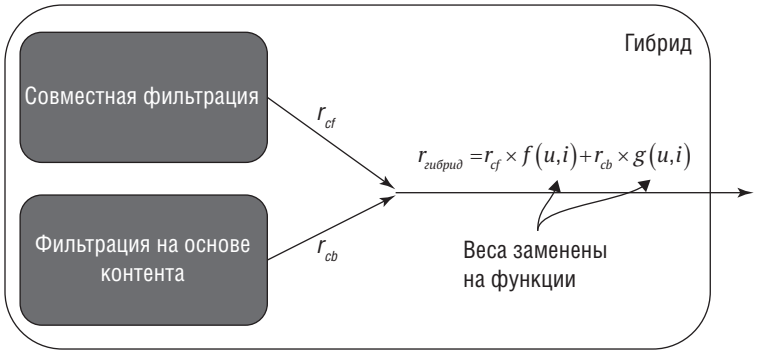


Рис. 12.10. Пример рекомендатора FWLS, который заменяет веса

Эти функции называются метафункциями, и тогда функция прогноза будет выглядеть следующим образом:

$$b(u,i) = f(u,i) \times r_{cf}(u,i) + g(u,i) \times r_{cb}(u,i).$$

На рис. 12.11 показан выбор метафункций, используемых на конкурсе Netflix Prize. Они кажутся простыми, и трудно поверить, что этого достаточно, чтобы почти выиграть миллион долларов. Эти функции заслуживают уважения!

Таблица 1: Метапризнаки, используемые для смешивания моделей на Netflix Prize	
1	Постоянная функция голосования (это позволяет строить регрессию исходных предикторов в дополнение к их взаимодействию с функциями голосования)
2	Бинарная переменная, обозначающая, оценил ли пользователь в конкретный день более трех фильмов
3	Журнал количества оценок фильма
4	Журнал количества различных дат, в которые пользователь оценивал фильмы
5	Байесовская оценка средней оценки фильма после вычитания байесовского среднего для заданного пользователя
6	Журнал количества оценок пользователей
7	Средняя оценка пользователя, байесовским способом приведенная к среднему значению для всех пользователей

Рис. 12.11. Весовые функции, используемые на конкурсе Netflix Prize

8	Норма вектора факторов SVD пользователя из 10-факторного SVD, обученного по остаточным значениям глобальных эффектов
9	Норма вектора факторов SVD фильма из 10-факторного SVD, обученного по остаточным значениям глобальных эффектов
10	Журнал суммы положительных корреляций оцененных фильмов и предсказываемого фильма
11	Стандартное отклонение прогноза 60-факторного порядкового SVD
12	Журнал среднего количества оценок у тех пользователей, которые оценили данный фильм
13	Журнал стандартного отклонения дат, в которые фильму ставилась оценка. Несколько оценок, поставленных в одну и ту же дату, попадают в расчет несколько раз
14	Процент величины корреляции в признаке 10 оценивается для 20 % фильмов с наилучшей корреляцией из тех, что оценил пользователь
15	Стандартное отклонение среднего для пользователя на конкретную дату от модели, где пользовательское среднее отделяется для каждой даты
16	Стандартное отклонение пользовательских оценок
17	Стандартное отклонение оценок фильмов
18	Журнал (дата оценки - дата первой оценки пользователя + 1)
19	Журнал количества оценок пользователя на дату + 1
20	Максимальная корреляция фильма с любым другим фильмом независимо от того, были ли пользователем оценены другие фильмы
21	Признак 3, умноженный на признак 6, т. е. журнал количества оценок пользователей, умноженный на журнал количества оценок фильмов
22	Среди пар пользователей, оценивших фильм, среднее перекрытие в наборах фильмов, которые оценили два пользователя, где перекрытие рассчитывается как процентное соотношение фильмов в меньшем из двух наборов, которые также находятся в большем из двух наборов
23	Процент оценок фильма, которые были единственной оценкой в течение дня для пользователя
24	(Нормализованное) Среднее количество оценок фильмов для фильмов, оцененных пользователем
25	Сначала создается матрица фильм–фильм с записями, в которых указана вероятность того, что пара фильмов была оценена в один и тот же день при условии, что пользователь оценил оба фильма. Затем для каждого фильма вычисляется корреляция между этим вектором вероятности по всем фильмам и вектором корреляций оценок по всем фильмам

Рис. 12.11. *Окончание*[1]

[1] Найдено в статье «Функциональное-взвешенное линейное сочетание» Дж. Силла. Доступно в интернете по адресу **arxiv.org/pdf/0911.0460.pdf**.

12.5.2. Алгоритм

Если у вас есть функция, показанная в предыдущем разделе, – это хорошо, но как определить, как она должна выглядеть? Список функций, показанный на рис. 12.11, – это результат хорошего знания данных, но и без «пальца в небо» тоже не обошлось.

Положим, что у вас есть ряд рекомендаторов, g_1, g_2, ..., g_t. Каждый g_i принимает на вход пользователя и элемент и возвращает прогноз оценки. Для этого вы можете написать простую линейную функцию:

$$b_{FW}(u,i) = w_1 g_1(u,i) + w_2 g_3(u,i) + \ldots + w_L g_L(u,i).$$

Это называется FW – функциональное взвешивание. Как мы видели в предыдущих главах, существует более простой способ записи:

$$b_{FW}(u,i) = \sum_{j=1}^{L} w_j g_j(u,i).$$

Выглядит слишком просто, не так ли? Представим каждый вес в виде функции, как мы видели ранее. Получим признако-взвешенные функции:

$$b_{FW}(u,i) = \sum_{j=1}^{L} w_j(u,i) g_j(u,i).$$

Теперь у вас есть быстрый способ записи линейных функций, и можно добавлять крутые примочки. Наши взвешенные функции могут быть определены следующим образом: каждый w_j определяется как сумма функций с весом v, как показано в следующей формуле:

$$w_j(u,i) = \sum_{k=1}^{M} v_{kj} f_k(u,i).$$

Странновато, не так ли? По крайней мере, я так и подумал в первый раз, когда увидел это. Каждая весовая функция является суммой всех метафункций. Идея заключается в том, чтобы машина сама решила, что лучше. Вы можете, например, сказать, чтобы рекомендатор на основе контента давал 90 % рекомендаций, если пользователь оценил менее трех элементов. В противном случае пусть будет деление 50/50. Давайте рассмотрим пример. У вас может быть две функции, такие как функции 1 и 2, показанные далее. Первая функция может быть такой:

$$f_1(u,i) = 1,$$

а вторая:

$$f_2(u,i) = \begin{cases} 1, \text{если } u \text{ оценил менее 3 элементов} \\ 0 \text{ в противном случае} \end{cases}.$$

У вас было два предиктора для совместной фильтрации и для фильтрации на основе контента. Вы можете сделать смешанную функцию b следующим образом:

$$b(u,i) = \left(v_{11} \times f_1(u,i) + v_{12} \times f_2(u,i)\right) r_{cf}(u,i) + \left(v_{21} \times f_1(u,i) + v_{22} \times f_2(u,i)\right) r_{cb}(u,i).$$

Это уже длинновато. Угадайте, какие значения будут у *v*? Если вы установите такими, как показано дальше, вы получите что хотели:

$$b(u,i) = \left(0,5 \times f_1(u,i) + -0,4 \times f_2(u,i)\right) p_{cf}(u,i) + \left(0,5 \times f_1(u,i) + 0,4 \times f_2(u,i)\right) p_{cb}(u,i).$$

Если пользователь оценил более трех элементов, то $f_2(u,i) = 0$ и $f_1(u,i) = 1$,

$$b(u,i) = \left(0,5 \times 1 + (-0,4) \times 0\right) p_{cf}(u,i) + \left(0,5 \times 1 + 0,4 \times 0\right) p_{cb}(u,i) = 0,5 \times p_{cf} + 0,5 \times p_{cb}.$$

Если пользователь оценил менее трех пунктов, то $f_2(u,i) = 1$ и $f_1(u,i) = 1$,

$$b(u,i) = \left(0,5 \times 1 + (-0,4) \times 1\right) p_{cf}(u,i) + \left(0,5 \times 1 + 0,4 \times 1\right) p_{cf}(u,i) = 0,1 \times p_{cf} + 0,9 \times p_{cb}.$$

Как раз то, что мы хотели. Чтобы вернуться в режим теории, давайте рассмотрим предыдущие выражения и объединим их для получения более компактного выражения r_{FWLS}:

$$r_{FW}(u,i) = \sum_{j=1}^{L} w_j g_j(u,i)$$

$$r_{FWLS}(u,i) = \sum_{j=1}^{L} \left(\sum_{k=1}^{M} v_{kj} f_k(u,i) \right) g_j(u,i)$$

$$w_j(u,i) = \sum_{k=1}^{M} v_{kj} f_k(u,i).$$

Теперь у вас есть FWLS, и это показано на рис. 12.12. Вы можете использовать это, чтобы смешать результаты рекомендаторов, используя веса, которые являются функциями. Все это дает вам большое преимущество.

Ручная установка значений – это неплохо, но нам нужно, чтобы машина, глядя на данные, сама решила, какие значения лучше. Значит, нам нужно обучить алгоритм. Давайте называть вещи своими именами – нам нужно машинное обучение.

Обучение алгоритма сводится к примерно такой же задаче, которую вы видели в предыдущей главе. Нам нужно взять данные, которые есть в вашей базе данных, и использовать их, чтобы определить, какими должны быть веса. В качестве примера положим, что у вас есть обычная матрица оценок, показанная в табл. 12.4.

Поскольку у нас шесть пользователей и шесть элементов, получаем 6 × 6 – 3 (минус три пустые ячейки в таблице), т. е. 33 точки данных, каждая из которых содержит ID пользователя, ID элемента и оценку (например, Сара, «Стартрек», 3). Будем считать данные правильными и скажем, если мы вводим (Сара, «Стартрек») в функцию, мы хотим на выходе увидеть значение 3. То же самое справедливо и для всех остальных 32 точек данных.

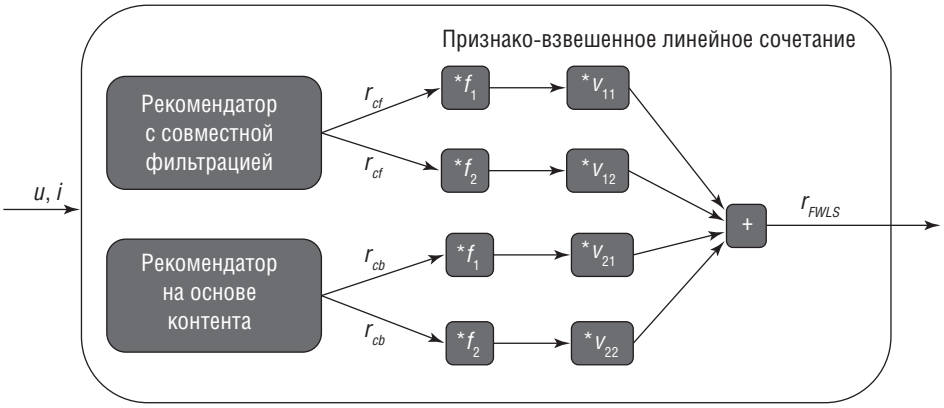


Рис. 12.12. Пример рекомендатора FWLS, содержащего рекомендаторы с совместной фильтрацией и на основе контента.

Таблица 12.4. Пример матрицы оценок

						
	Комедия	**Боевик**	**Комедия**	**Боевик**	**Драма**	**Драма**
Сара	5	3		2	2	2
Джаспер	4	3	4		3	3
Тереза	5	2	5	2	1	1
Хелла	3	5	3		1	1
Петро	3	3	3	2	4	5
Екатерина	2	3	2	3	5	5

Мы хотим, чтобы функция гибридного рекомендатора давала такой выход, в котором разница между выражением и фактическими оценками была как можно меньше. В примере (Сара, «Стартрек») требуется свести к минимуму следующее:

$$r_{FSWL}(Сара,\ Стартрек) - 3.$$

Требуется сделать кое-что еще, потому что, если мы установим оценку 0 в предыдущем выражении, результат будет равен –3, т. е. минимальный, который вы можете получить. Давайте вместо этого попробуем минимизировать следующее:

$$\left(r_{FSWL}(Сара,\ Стартрек) - 3\right)^2 \quad (1),$$

поскольку в этом случае получится придерживаться цели, двигаясь как можно ближе к нулю. Выражение (1) – это только для одной из 33 точек данных, которые у вас есть. И вы должны быть уверены в том, что функция работает для них всех, так что это надо будет сделать для всех пользователей и элементов. Чтобы сделать это, нужно следующее:

$$\sum_{u \in \text{пользователей}} \sum_{u \in \text{элементов}} \left(r_{FSWL}(u,i) - r_{u,i}\right)^2.$$

Что такое r_{FSWL}? В следующем выражении вместо r_{FSWL} подставляется выражение:

$$\sum_{u \in \text{пользователей}} \sum_{u \in \text{элементов}} \left(\sum_{j=1}^{L} \sum_{k=1}^{M} v_{kj} f_k(u,i) g_j(u,i) - r_{u,i} \right).$$

И вот это нам нужно минимизировать. Это наш алгоритм.

У нас много различных способов взяться за это. Закончим пример с вручную найденными весами, которые дали нам следующую функцию:

$$b(u,i) = \left(0{,}5 \times f_1(u,i) + -0{,}4 \times f_2(u,i)\right) p_{cf}(u,i) + \left(0{,}5 \times f_1(u,i) + 0{,}4 \times f_2(u,i)\right) p_{cb}(u,i).$$

Если вы хотите знать, как рекомендатор предскажет оценку Сары для фильма «Мстители», нужно вызвать оба функциональных рекомендатора (предположим, что они предсказывают 4 и 5 соответственно) и запустить обе функции. Обе возвращают 1. То есть гибрид считает как-то так:

$$b(u,i) = \left(0{,}5 \times 1 + \left(-0{,}4\right) \times 1\right) \times 4 + \left(0{,}5 \times 1 + 0{,}4 \times 1\right) \times 5 = 4{,}9.$$

Теперь давайте вернемся к весам и посмотрим, как научиться решать эту проблему.

Когда рекомендатор не справляется

Нам нужно подумать о том, что делать, когда один или несколько из рекомендаторов не могут дать каких-либо рекомендаций. Проще всего удалить из набора данные, на которые один из рекомендаторов не отвечает. Но это уменьшает обучающий набор, так что решение сомнительное. Вы также можете попробовать угадать, добавив среднюю оценку пользователя, элемента, среднее из двух или базисный прогноз, описанный в главе 11.

В следующей реализации выполнено первое решение – удалены все строки, в которых нет прогнозов оценок от какого-либо из функциональных рекомендаторов. Также возможно, что функциональное взвешивание не определено для всех строк; в этом случае вы также можете удалить строку или придумать значение по умолчанию.

Обучение

Для обучения гибридного рекомендатора вам нужно сделать несколько вещей. Длинная процедура, показанная на рис. 12.13, может привести к пута-

нице и превратить маленькие ошибки в большие. Мы выполним следующие шаги:

- разделим данные на два набора: обучающие и тестовые (не показано на рис. 12.13);
- обучим функциональные рекомендаторы;
- сформируем прогнозы для каждой из точек обучающих данных;
- выполним каждую взвешенную функцию на всех учебных и обычных данных;
- для каждой точки обучающих данных вычислим произведение между каждой спрогнозированной оценкий и результатом функционально взвешенной функции;
- найдем неизвестные значения с помощью линейной регрессии;
- протестируем гибрид на тестовом наборе.

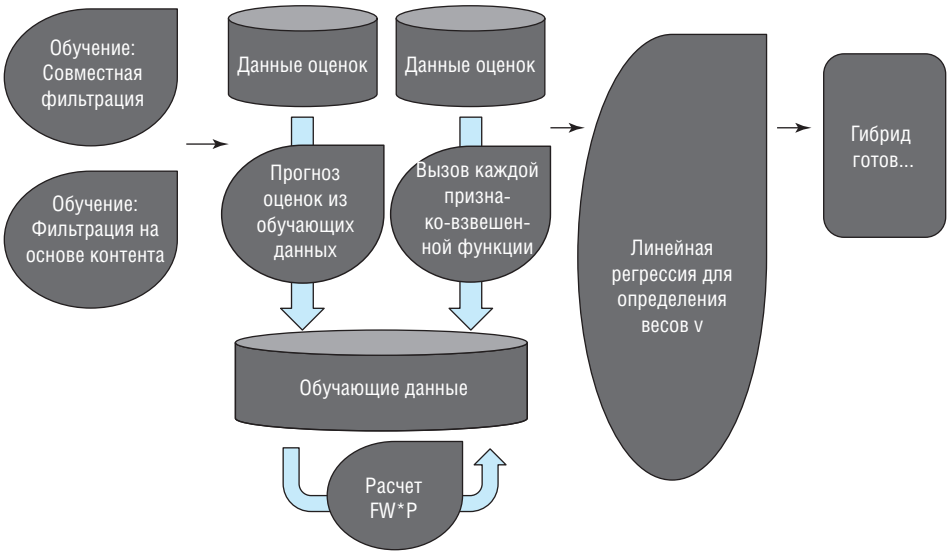


Рис. 12.13. Обучение гибридного рекомендатора FWLS

Разделение данных

Перед началом стоит разделить данные и удалить порядка 20 % из них, чтобы проверить, насколько хорошо работает алгоритм. Тут идея заключается в том, что если вы используете все данные для обучения алгоритма, то его эффективность будет измеряться на данных, которые он уже видел. Поэтому вам нужно убрать часть данных, чтобы можно было видеть, насколько хорошо он будет работать с незнакомыми данными. И, как уже упоминалось в предыдущей главе, разделение данных перед обучением рекомендатора означает, что функциональный рекомендатор не будет обучаться на тестовых данных.

Чтобы сделать это правильно, вы должны разделить ваши данные на обучающие, проверочные и тестовые. Для обучения функциональных рекомендаторов и гибрида используйте только обучающие данные. Затем используйте

проверочные данные, чтобы оценить качество модели гибрида и выполнить тонкую настройку гиперпараметров. Тестовый набор используется для объективной проверку модели.

Обучение функциональных рекомендаторов

Здесь нам потребуются знания, полученные в предыдущих главах. Перед созданием гибрада нужно подготовить рекомендаторы, которые войдут в его состав. Во-первых, подготовим рекомендатор с совместной фильтрацией элемент–элемент (который был описан в главе 8), а также рекомендатор на основе контента (глава 10). Можно использовать и другие виды, но я выбрал эти.

Генерации прогнозов для каждой точки обучающих данных

Здесь мы столкнемся со странной проблемой: как рекомендатору реагировать на ситуацию, когда вы пытаетесь предсказать оценки, которые уже существуют? В зависимости от реализации ваша система совместной фильтрации может решить использовать элемент, наиболее подобный текущему элементу. В этом случае вместо предсказания рекомендатор вернет реальную оценку. Это не очень хорошая идея, потому что это затруднит обучение гибрида.

Здесь мы решим эту проблему, представив, что элемент текущим пользователем оценен не был. В табл. 12.5 показаны реальные оценки.

Таблица 12.5. Обучающие данные для «Людей в черном»

Пользователь	Элемент	Оценка
Сара	ЛВЧ	5
Джаспер	ЛВЧ	4
Тереза	ЛВЧ	5
Хелла	ЛВЧ	4

Когда рекомендатор будет обучен и готов пойти в бой, вы берете обучающие данные, которые будете использовать для обучения гибрида, и запускаете их на каждом из рекомендаторов, а затем посмотрите на результат. Это дает вам данные, приведенные в табл. 12.6.

Таблица 12.6. Обучающие данные с прогнозами

Пользователь	Элемент	Прогноз совместной фильтрации	Прогноз на основе контента	Реальная оценка
Сара	ЛВЧ	4,5	3,5	5
Джаспер	ЛВЧ	4	5	4
Тереза	ЛВЧ	4	4	5
Хелла	ЛВЧ	3	5	4

Выполнение каждой признако-взвешенной функции на всех обучающих данных

Теперь мы пропустим данные через две функции, чтобы получить их результаты. Это показано в табл. 12.7.

Таблица 12.7. Обучающие данные с прогнозами и результатами функций

Пользователь	Элемент	Прогноз совместной фильтрации	Прогноз на основе контента	F1	F2	Реальная оценка
Сара	ЛВЧ	4,5	3,5	1	0	5
Джаспер	ЛВЧ	4	5	1	1	4
Тереза	ЛВЧ	4	4	1	1	5
Хелла	ЛВЧ	3	5	1	1	4

Для каждой точки обучающих данных вычисляется произведение между каждой прогнозируемой рейтинга и каждым результатом взвешенной функции, а затем прогноз умножается на метафункцию. Вы получите результаты, показанные в табл. 12.8.

Таблица 12.8. Обучающие данные с расчетными прогнозами и результатами функций

Пользователь	Элемент	Pcf × F1	Pcf × F2	Pcb × F1	Pcb × F2	Реальная оценка
Сара	ЛВЧ	4,5	0	3,5	0	5
Джаспер	ЛВЧ	4	4	5	5	4
Тереза	ЛВЧ	4	4	4	4	5
Хелла	ЛВЧ	3	5	3	5	4

Теперь обучающие данные готовы.

Поиск неизвестного значения против линейной регрессии

Для того чтобы найти неизвестные *v*, мы воспользуемся линейной регрессией. Это означает, что мы попытаемся найти функцию, которая создаст максимально близкий к фактическим данным выходной сигнал. Такая функция не всегда может быть найдена, но мы вернемся к этому чуть позже.

Линейная регрессия – это одна из первых вещей, которую нужно освоить в контексте машинного обучения. На эту тему были написаны целые книги, поэтому здесь мы останавливаться на этом не будем[1]. Отмечу, что идея тут в основном та же, как и в предыдущей главе, когда вы пытались найти неизвестные в Funk SVD, в том смысле, что линейная регрессия будет пытаться минимизировать квадратичную ошибку по всем точкам данных.

[1] G. James et al., «Introduction to Statistical Learning» (Springer, 2017). Эта книга не совсем о линейной регрессии, но ей посвящена целая глава.

Тестирование гибрида на тестовом наборе

Выполнив линейную регрессию, вы можете проверить качество результатов, выдаваемых гибридом (по сравнению с оценками тестового набора, которые мы исключили) и решить, устраивает ли вас его работа

Чтобы сделать это, мы исследуем, насколько близки предсказания гибрида к реальным оценкам. Теперь, когда вы знаете все это, пришло время взглянуть на код, реализующий ваш гибрид.

12.6. Реализация

Давайте посмотрим, как гибрид FWLS работает на сайте MovieGEEKs. Мы выполним те же действия, что и ранее, но уже в виде кода. Если вы еще не загрузили код MovieGEEKs, предлагаю делать это сейчас, чтобы вы не отставали от происходящего. Подготовка MovieGEEKs к работе выполняется несколько этапов. Вы можете найти код по ссылке **mng.bz/ 04k5**. Загрузив код, следуйте инструкциям по установке, приведенным в файле *Readme*.

Во время установки вы загрузите набор данных MovieTweetings. Если вы выполнили инструкции и загрузили все данные, сможете обучить гибридный алгоритм FWLS, запустив сценарий ниже:

ЛИСТИНГ 12.1. Обучение рекомендатора FWLS

```
> python -m builder.fwls_calculator
```

Работу сценария можно видеть в реальном времени. Он начинает загрузку и разделение данных.

Загрузка данных

Теперь мы загрузми данные оценок (я хочу убедиться, что вы понимаете, что это оценки, полученные от пользователей, явные или неявные). Эти данные используются для тонкой настройки гибрида – т. е. вход и выход (ожидаемый). Вы найдете следующие сценарии в файле */builder/fwls_calculator.py*.

ЛИСТИНГ 12.2. Обучение рекомендатора FWLS

```
def get_real_training_data(self):
    columns = ['user_id', 'movie_id', 'rating', 'type']
    ratings_data = Rating.objects.all().values(*columns)
    df = pd.DataFrame.from_records(ratings_data, columns=columns)
```

Созд ние П нды и д т фрейм

Чтобы получить данные, вы загрузили все оценки в память. Если вы видите красные флажки – то так и должно быть.

Нежелательно загружать всю таблицу в память, если вы не ищете неприятностей. Но здесь данных не так много. В противном случае нужно либо организовывать поток данных по кускам, либо брать выборку данных. Проще

всего было бы просто отсечь старые данные до момента, пока не останется нормальный объем. Двигаемся дальше.

Разделение данных

Здесь вы разделим данные на два набора: тестовые и обучающие. Это делается, чтобы вы могли оценить результат нашей работы. Вы будете использовать эти данные для обучения функциональных рекомендаторов и гибридного рекомендатора. Для гибрида мы будем использовать обучающие данные, чтобы выбрать подходящие значения для весов, а затем проверим все это.

ЛИСТИНГ 12.3. Разделение данных на обучающие и тестовые

```
self.train, self.test = train_test_split(self.train, test_size = 0.2)
```

Обучение функциональных рекомендаторов

Функциональные рекомендаторы нужно обучать, но мы уже обучали их в предыдущих главах. Но если нужно оценивать алгоритмы, вы должны обучать их только на обучающих данных. Во всяком случае для совместной фильтрации это так. Рекомендатор на основе контента использует данные контента, поэтому отсечение лишних пользователей ничего не изменит. Чтобы обучить функциональные рекомендаторы, обновите строку в следующем листинге.

ЛИСТИНГ 12.4. Вызов обучающего метода

```
fwls.train() # run fwls.train(train_feature_recs=True)
to train feature recs
```

Генерация прогнозов для каждой точки обучающих данных

Загрузив оценки, вы можете взять каждую из точек данных в обучающих данных и увидеть, что предсказывает каждый из функциональных рекомендаторов. Прогнозы добавятся в данные.

Вот вызываемые методы будут делать прогноз. Первый – это совместная фильтрация, которую мы подробно описали в главе 8 (обратитесь к этой главе, если подзабыли). Код показан в листинге 12.5. Обратите внимание, что метод принимает список `movie_ids`. Это делается для того, чтобы не использовать данные о том, что пользователь уже оценил элемент. Метод вызывается со списком всех элементов, оцененных пользователем, кроме элемента, который вы хотите спрогнозировать. Вы найдете следующие сценарии в файле */recs/neighbourhood_based_recommender.py*.

ЛИСТИНГ 12.5. Совместная фильтрация

```
def predict_score_by_ratings(self, item_id, movie_ids):
    top = 0
    bottom = 0
    mc = self.max_candidates
```

```
    ids= movie_ids.keys()
    candidate_items = (Similarity.objects.filter(source__in=ids)
                                         .filter(target=item_id)
                                         .exclude(source=item_id)
    candidate_items = candidate_items.distinct()
                                      .order_by('-similarity')[:mc]

    if len(candidate_items) == 0:
        return 0

    for sim_item in candidate_items:
        r = movie_ids[sim_item.source]
        top += sim_item.similarity * r
        bottom += sim_item.similarity

    return top/bottom
```

Отсечение имеющихся оценок пользов теля

Второй рекомендатор (листинг 12.6) – это рекомендатор на основе контента с моделью LDA. Модель LDA была подробно описана в главе 10, но этого метода там не было. Здесь мы используем значения сходства и оценки пользователя, чтобы спрогнозировать оценку. При этом сходство берется из модели LDA, а не из поведения пользователей, как в совместной фильтрации.

ЛИСТИНГ 12.6. Рекомендатор на основе контента

```
def predict_score(self, user_id, item_id):

    active_user_items = (Rating.objects.filter(user_id=user_id)
                         .exclude(movie_id=item_id)
                                  .order_by('-rating')[:100])

    movie_ids = {movie['movie_id']: movie['rating'] for movie in
     active_user_items}
    user_mean = sum(movie_ids.values()) / len(movie_ids)

    sims = LdaSimilarity.objects.filter(Q(source__in=movie_ids.keys())
                                    & Q(target=item_id)
                                    & Q(similarity__gt=self.min_sim)
                                    ).order_by('-similarity')
    pre = 0
    sim_sum = 0

    if len(sims) > 0:
        for sim_item in sims:
            r = Decimal(movie_ids[sim_item.source] - user_mean)
            pre += sim_item.similarity * r
            sim_sum += sim_item.similarity

    return Decimal(user_mean) + pre / sim_sum
```

Мы будем использовать эти два метода на каждой оценке в обучающих данных, применяя их к каждой строке и вставляя результаты в новые столбцы, как показано в следующем листинге. Сценарий находится в файле */builder/fwls_calculator.py/*.

ЛИСТИНГ 12.7. Расчет прогнозов для обучающих данных

```
def calculate_predictions_for_training_data(self):

    self.training_data['cb'] = self.training_data.apply(lambda data: self.cb.predict_
score(data['user_id'], data['movie_id']))
    self.training_data['cf'] = self.training_data.apply(lambda data: self.cf.predict_
score(data['user_id'], data['movie_id']))
```

Теперь у вас есть структура данных, показанная в табл. 12.2.

Применение каждой признако-взвешенной фукнции ко всем точкам обучающих данных

Функции, которые вы будете использовать на этом этапе, довольно просты, и обсуждать тут особо нечего, но их вполне достаточно, чтобы показать, как работает FWLS, – см. листинг ниже. Вы найдете этот сценарий в файле */builder/fwls_calculator.py/*.

ЛИСТИНГ 12.8. Иллюстрация работы FWLS

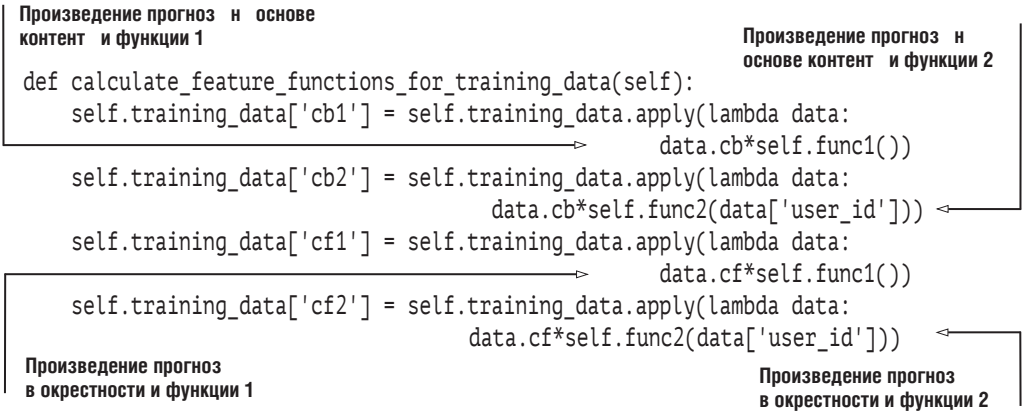
```
def fun1(self):                    <-- Всегд  возвр  щ  ет 1
    return 1.0

                                       Возвр  щ  ет 1, если пользов  тель оце-
                                       нил более трех фильмов, в противном
                                       случ  е возвр  щ  ет 0
def fun2(self, user_id):           <--
    user_ratings = self.rating_count['user_id']==user_id
    rating_count = self.rating_count[user_ratings]['movie_id'].values[0]
    if rating_count < 3.0:
        return 1.0
    return 0.0
```

Вместо вычисления функции и сохранения данных мы будем использовать функции и сразу вычислим произведение их результатов.

Для каждой точки обучающих данных вычисляем произведения всех прогнозов оценкой для каждой функции

Для каждого прогноза мы получим столбец с произведениями результатов каждой из функций. У вас есть два предиктора и две функции, т. е. у нас получится четыре новых столбца, как показано в следующем листинге. Сценарий опять же в файле */builder/fwls_calculator.py/*.

ЛИСТИНГ 12.9. Расчет функций для обучающих данных



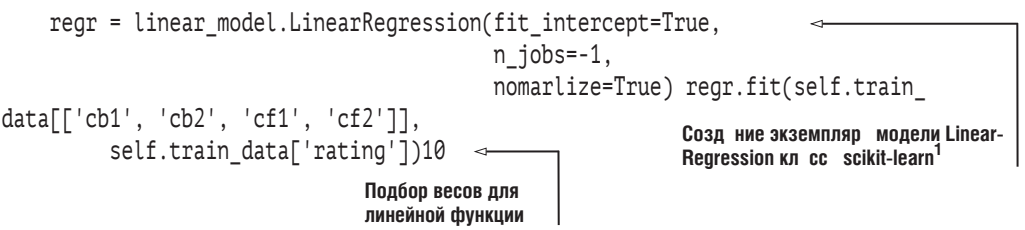
Мы закончили подготовку к выполнению линейной регрессии и готовы найти значения *v*. Это также называется *генерацией признаков*. Часто при выполнении приложений машинного обучения именно эта часть занимает большую часть времени.

Результат взвешенных функций

Мы рассчитали признако-взвешенные функции, но в реальной жизни это не всегда стоит выполнять. Иногда лучше делать маленькие шаги в зависимости от конкретной задачи.

Перейдем к линейной регрессии. Вы найдете эти сценарии в методе обучения в файле */builder/fwls_calculator.py*.

ЛИСТИНГ 12.10. Линейная регрессия



Чтобы использовать эти веса, нужно сохранить их. Вы можете поместить их в базу данны для более позднего использования или сохранить их в файл, как показано в следующем листинге.

[1] Для дополнительной информации см. **mng.bz/P375.**

ЛИСТИНГ 12.11. Сохранение весов

```
with open(self.save_path + 'fwls_parameters.data', 'wb') as ub_file:
        pickle.dump(result, ub_file)
```

Прогнозирование рекомендаций онлайн

Вы можете реализовать гибридный рекомендатор, выполнив предварительный расчет рекомендаций и сохранив их в базу данных. Более ленивый подход – это вызывать все функции рекомендаторов и смешивать результаты.

Чтобы правильно отсортировать результаты, оба рекомендатора могут дать топ-*N* в пять раз больше, чем запрашивает пользователь. Это должно позволить линейным функциям иметь достаточное количество элементов, чтобы правильно отсортировать их, как показано в следующем листинге. Вы найдете эти сценарии в методе обучения в файле */builder/fwls_calculator.py*.

ЛИСТИНГ 12.12. Сортировка элементов

```
def recommend_items(self, user_id, num=6):
    cb_recs = self.cb.recommend_items(user_id, num * 5)
    cf_recs = self.cf.recommend_items(user_id, num * 5)

    combined_recs = dict()
    for rec in cb_recs:
        movie_id = rec[0]
        pred = rec[1]['prediction']
        combined_recs[movie_id] = {'cb': pred}

    for rec in cf_recs:
        movie_id = rec[0]
        pred = rec[1]['prediction']
        if movie_id in combined_recs.keys():
            combined_recs[movie_id]['cf'] = pred
        else:
            combined_recs[movie_id] = {'cf': pred}
    fwls_preds = dict()
    for key, recs in combined_recs.items():
        if 'cb' not in recs.keys():
            recs['cb'] = self.cb.predict_score(user_id, key)
        if 'cf' not in recs.keys():
            recs['cf'] = self.cf.predict_score(user_id, key)
        pred = self.prediction(recs['cb'], recs['cf'], user_id)
        fwls_preds[key] = {'prediction': pred}
sorted_items = sorted(fwls_preds.items(),
key=lambda item: -float(item[1]['prediction']))[:num]

 return sorted_items
```

Вызов рекоменд тор н основе контент с з просом пятикр тного числ элементов

Вызов рекоменд тор в окрестности с з просом пятикр тного числ элементов

Проход через все рекоменд ции и созд ние слов ря слов рей. Доб вление всех элементов из рекоменд ций н основе контент

Доб вление всех элементов из модели окрестности

Проход через модель и получение прогнозов недост ющих оценок от рекоменд торов

Р счет прогнозов оценок для всех элементов в сло в ре с помощью метод , пок з нного в листинге 12.13

Сортировк результ т в прогнозе

Метод прогнозирования выглядит так, как показано в следующем листинге.

ЛИСТИНГ 12.13. Метод прогнозирования

```
def prediction(self, p_cb, p_cf, user_id):
    p = (self.wcb1 * self.fun1() * p_cb +
         self.wcb2 * self.fun2(user_id) * p_cb +
         self.wcf1 * self.fun1() * p_cf +
         self.wcf2 * self.fun2(user_id) * p_cf)
    return p + self.intercept

def fun1(self):
    return Decimal(1.0)

def fun2(self, user_id):
    count = Rating.objects.filter(user_id=user_id).count()
    if count > 3.0:
        return Decimal(1.0)
    return Decimal(0.0)
```

Р ссчитыв ет прогнозируемый рейтинг для элемент

Функция 1

Функция 1

В следующем разделе вы увидите несколько рекомендаций, полученных от гибрида.

Как гибрид выполняет сравнение?

Результаты функциональных рекомендаторов коррелируют с гибридным рекомендатором, но как гибрид сравнивается с отдельными рекомендаторами? Чтобы понять это, посмотрите, какой из трех рекомендаторов (с совместной фильтрацией или на основе контента) ближе всего к реальным оценкам.

В первом примере, где у пользователя всего одна оценка, модель окрестностей ничего не возвращает (рис. 12.14). Затем, если вы посмотрите на пользователя, который поставил несколько оценок, сочетание двух рекомендаторов дает элементы, которые не войдут в топ-6 обоих составных рекомендаторов (рис 12.15).

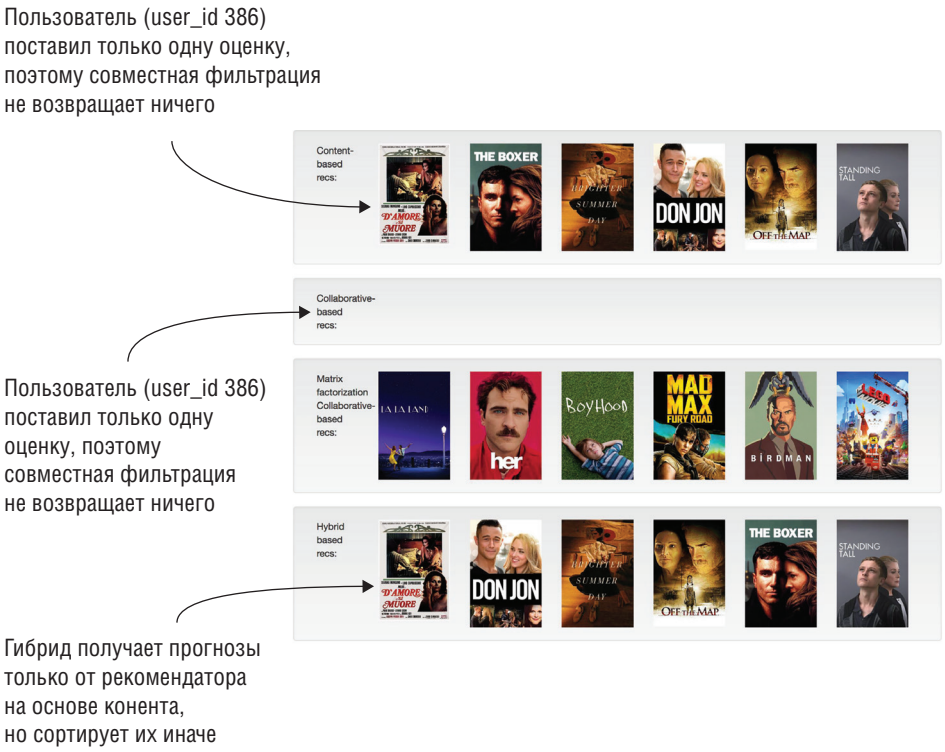


Рис. 12.14 Пример рекомендаций, возвращенных гибридным рекомендатором

Рис. 12.15. Рекомендации гибрида, полученные из результатов обоих рекомендаторов

Тестирование гибрида на тестовом наборе

Сработало? А как проверить? Есть офлайн-оценка, о которой мы узнали в главе 9. В ней выполняется перекрестная проверка, как показано на рис. 12.16.

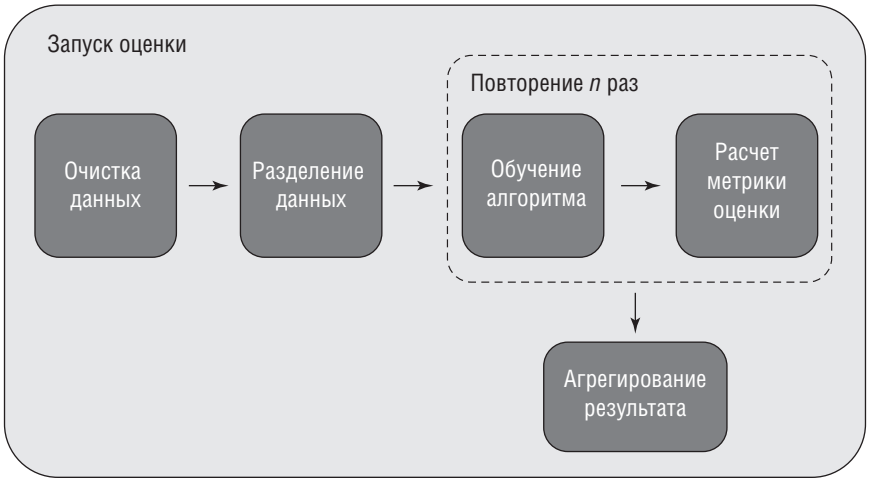


Рис. 12.16. Оценщик выполняет оценку алгоритма. Здесь данные сначала очищаются, а затем разделяются на *k* частей для перекрестной проверки. Тогда для каждой части выполняется подготовка алгоритма для оценки. Когда все закончится, вы агрегируете результат

Выполнение этой оценки похоже на то, что мы описывали в главе 9, только здесь нужно обучить все включенные рекомендаторы. Вы можете запустить оценку, выполнив код, приведенный в следующем листинге.

ЛИСТИНГ 12.14. Выполнение оценки

```
> python -m evaluator.evaluation_runner -fwls
```

Оценщик получает данные, показанные на рис. 12.17. Эта оценка (как изображено на рис. 12.17) отличается от других методов, которые вы видели, потому что она построена на трех алгоритмах машинного обучения. Но это не тот случай, когда конечный продукт лучше, даже если два функциональных рекомендатора оптимальны.

Если бы я хотел поставить эту систему в производство, то я бы, наверное, заранее рассчитал часть алгоритмов для ускорения процесса. Чтобы сделать обучение быстрее, можно взять часть оценок, используемых для выполнения линейной регрессии, как я сделал в листинге ниже.

ЛИСТИНГ 12.15. Выполнение оценки

```
self.train_data = self.train_data.sample(self.data_size)
```

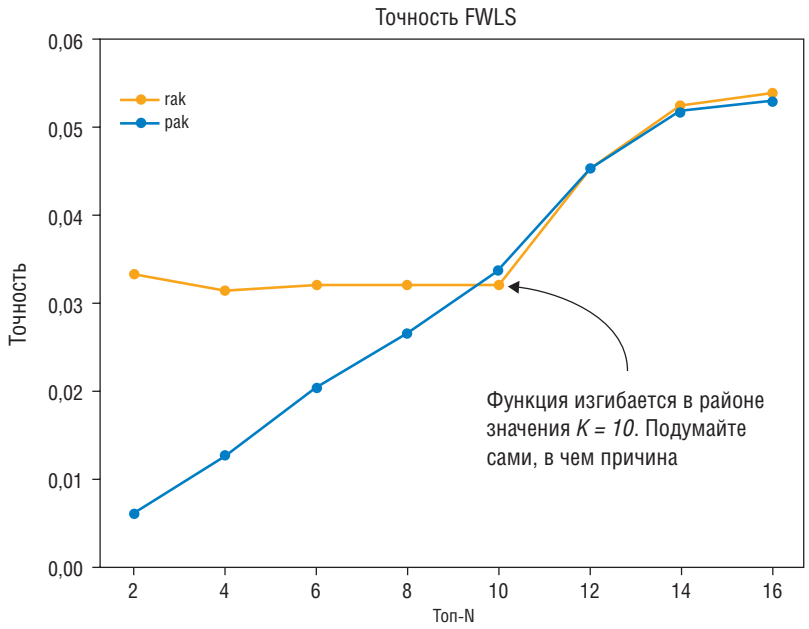


Рис. 12.17. Оценка алгоритма FWLS

Вопрос в том, насколько велик должен быть размер выборки, чтобы она была репрезентативна, учитывая, что мы обучаем четыре значения веса. Это типичная дилемма машинного обучения. Я могу сказать, что у меня все неплохо получилось с несколькими сотнями точек данных, но это не мало! Однако если вы посмотрите на точность в графике на рис. 12.17, то алгоритм выглядит хорошо.

Мы рассмотрели в этой главе много терминов и комбинаций. Если вы новичок в линейной регрессии, то я настоятельно рекомендую окунуться в нее поглубже. Потом все станет проще и вы сможете использовать ее для многих вещей, и отказываться от такого инструмента глупо.

Резюме

- Рекомендательная система может быть значительно оптимизирована путем совмещения результатов нескольких алгоритмов.
- Гибридные рекомендаторы позволяют объединить усилия различных рекомендаторов, чтобы получить лучшие результаты.
- В реальной жизни не все рабочие алгоритмы являются архисложными, и алгоритм, который выиграл премию Netflix, оказался непригоден для реальной работы.
- Признако-взвешенное линейное сочетание (FWLS) позволяет системе использовать разные рекомендаторы со своими весами, что добавляет мощи системе.

Глава 13

Ранжирование и обучение ранжированию

Эта книга об обучении, и в этой главе вы узнаете, как выполнять ранжирование:

- мы переформулируем задачу рекомендатора, превратив ее в задачу ранжирования;
- рассмотрим метод ранжирования Foursquare и то, как в нем используется несколько источников;
- рассмотрим различные алгоритмы обучения ранжированию (LTR) и научимся различать точечное, парное и списочное сравнения рангов;
- изучим алгоритм байесовского персонализированного ранжирования (БНР) – весьма перспективный алгоритм.

Главы про рекомендательные алгоритмы начинают выглядеть как-то одинаково, да? Теперь вам повезло, потому что мы возьмемся за нечто совершенно иное. Вместо того чтобы смореть на рекомендации как на задачу предсказания оценки, иногда имеет смысл взглянуть на то, как расставлять элементы. Самый релевантный элемент из каталога должен выводиться вверху списка, следующий – на второй позиции и т. д. Для того чтобы определить релевантность, нам не нужно прогнозировать оценки. Вам не нужно знать, как именно пользователь оценил элемент, – достаточно знать, что этот элемент ему нравится больше других.

ПРИМЕЧАНИЕ. Имейте в виду, что каталог контента может и не содержать элементов, которые понравятся пользователю, но даже в этом случае вам хочется выдать рекомендации как можно лучше.

Я думаю, это будет интересная глава. Вы узнаете об алгоритме, введенном в области информационного поиска (IR) – модное слово для поисковых систем. Система ранжирования поисковой системы Bing компании Microsoft, а также большинство других поисковых систем, Facebook и Foursquare, тоже используют его. Вы увидите разницу между тем, что находят они и что – ре-

комендатор. Однако большая часть исследований, проведенных в сфере IR, также подходит и для рекомендаторов.

Мы начнем путешествие с примера обучения ранжированию Foursquare, чтобы вы поняли саму идею ранжирования. Затем сделаем шаг назад и посмотрим на алгоритмы LTR в целом, которые разделены на три подгруппы. Чтобы привести конкретный пример алгоритмов LTR, мы исследуем алгоритм BPR. У него немного сложная математика, но мы закодируем это для MovieGEEKs, чтобы вы посмотрели на него в действии.

13.1. Обучение ранжированию на примере Foursquare

Foursquare – это путеводитель по городам. Я использую его, чтобы искать кофейни (обещаю себе, что слезу с кофеной иглы, когда допишу эту книгу). Представьте себе, что я стою перед красивым собором Святого Петра в Риме. После изнурительно долгого ожидания за право посмотреть красоты этой церкви, я решаю выпить кофе, достаю телефон, открываю приложение Foursquare и нажимаю на поиск кофе рядом со мной. Результат показан на рис. 13.1 (не полностью, но это браузерная версия Foursquare).

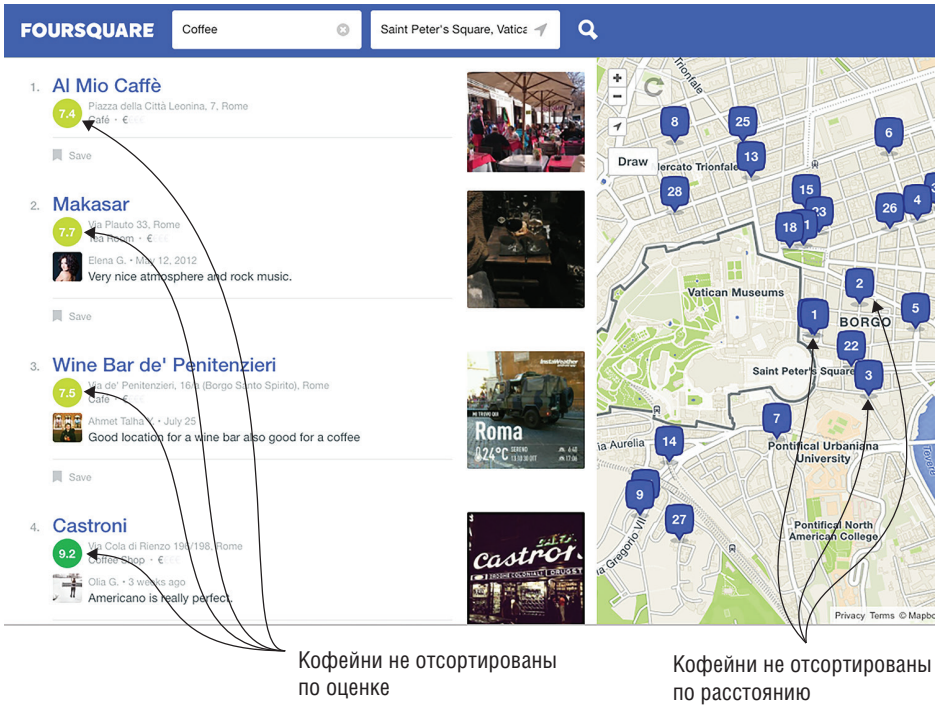


Рис. 13.1. Поиск коффеен возле собора Святого Петра в Риме через Foursquare

Как вы можете видеть, эти рекомендации не упорядочены ни по оценке, ни по расстоянию – но как тогда? Как Foursquare составил такой список?

В Foursquare опубликовали отличную статью о том, как работает их рекомендательная система, и описали их реализацию обучения ранжированию[1]. Я рекомендую вам прочитать ее, в ней очень интересно описаны проблемы в нахождении точек интереса вблизи пользователей. Мы рассмотрим несколько более простую версию обучения ранжированию в качестве вводного примера.

Как и гибридные рекоменаторы из главы 12, обучение ранжированию является способом сочетать различные виды источников данных, например популярность, расстояние, или выходные данные рекомендательной системы. Разница состоит в том, что ранг не обязательно исходит из рекомендательной системы. При ранжировании вы ищете источники информации, которые дадут вам принцип сортировки объектов. На рис. 13.2 показан список признаков, используемых в Foursquare.

Признак	Описание
Пространственный балл	$P(l \mid v)$
Временной фактор	$P(t \mid v)$
Популярность	Приближенная оценка посетителей в день
Находятся здесь	# Число пользователей в месте в данный момент
Личная история	# Число визитов от пользователя
Создатель	1, если пользователь создал место, 0 – если нет
Мэр	1, если пользователь – начальник места, 0 – если нет
Друзей здесь	# Число друзей в этом месте на момент запроса
История без/с учетом времени дня	# Число прошлых визитов в то же время дня

Рис. 13.2. Список признаков, используемых в алгоритме Foursquare для ранжирования мест рядом с вами

Поскольку у вас нет доступа к признакам, перечисленным на рис. 13.2, давайте оставим два из них и посмотрим, что можно понять из ранжирования, которое я привел на рис. 13.1. На странице показана средняя оценку каждого места. Расстояние я посмотрел с помощью Google Maps. Сведем эти данные в табл. 13.1.

Таблица 13.1. Рекомендации кофеен из Foursquare

Оценки на Foursquare	Название	Время ходьбы (расстояние)	Средняя оценка
1	Al Mio Caffé	2 мин	7,4
2	Makasar	4 мин	7,7
3	Wine bar be’	4 мин	7,5
	Penitenzieri		
4	Castroni	10 мин	9,2

[1] Блэки Шоу и др., «Обучение ранжированию для пространственно-временного поиска» **mng.bz/vP25**.

Из таблицы легко видеть, что ранжирование не основано на оценках, иначе Castroni была бы на самом верху. Расстояние тоже ни при чем, иначе бы Makasar делила второе место с винным баром. Давайте попробуем каким-то образом получить такое ранжирование, используя эти два признака.

Во-первых, вам нужно преобразовать данные таким образом, чтобы более высокое ранжирование означало меньшую дистанцию и большую оценку. Далекое расположение кафе – это плохо, так что придется обратить значения расстояний. То есть места с меньшим расстоянием будут давать более высокое значение. Чтобы сделать это, найдем максимум, который равен десяти минутам ходьбы, и вычтем каждое время для каждого кафе из максимума. Тогда для Al Mio Caffé получаем 10 – 2 = 8 и для Castroni 10 – 10 = 0. Масштабируем данные так, чтобы они лежали между 1 и 0, иначе некоторые алгоритмы могут не работать[1]. Для этого можно использовать следующую формулу:

$$x' = \frac{x - \min(x)}{\max(x) - \min(x)}.$$

Нормализация данных даст вам данные, показанные в табл. 13.2.

Таблица 13.2. То же самое, что в табл. 12.1, но с нормализованными данными

Оценка на Foursquare	Название	Время ходьбы (расстояние)	Средние оценки
1	Al Mio Caffé	1,00	0,00
2	Makasar	0,75	0,17
3	Wine bar de'	0,75	0,06
	Penitenzieri		
4	Castroni	0,00	1,00

Теперь сортировка по расстоянию уже близка к ранжированию на Foursquare. Поскольку элементы 2 и 3 имеют одинаковое время ходьбы, возьмем данные из оценок.

Мы подошли к сути проблемы: нужно научить машину ранжировать элементы на основании оценок и расстояния. Вы можете формализовать это, сказав, что система должна определить веса (w_0 и w_1), которые при вставке в следующее выражение дадут такое значение, что четыре элемента отсортируются так, как у Foursquare:

$$f(расстояние, оценка) = w_0 \times расстояние + w_1 \times оценка.$$

Так как вы хотите, чтобы функция сортировала данные, как Foursquare, попробуем создать алгоритм, который оптимизирован для ранжирования на основе выходных данных.

[1] Подробнее о масштабировании, см. **en.wikipedia.org/wiki/Feature_scaling**.

В этом примере не слишком трудно угадать значения w_0 и w_1. Если вы зададите $w_0 = 20$ и и $w_1 = 10$, вы получите значения, приведенные в табл. 13.3.

Таблица 13.3. Данные кафе с баллами, рассчитанными предыдущей функцией *f*

Оценка на Foursquare	Название	Время ходьбы (расстояние)	Средние оценки	Баллы
1	Al Mio Caffé	1	0	20
2	Makasar	0,75	0,17	16,7
3	Wine Bar de' Penitenzieri	0,75	0,06	15,6
4	Castroni	0	1	10

Другой способ подойти к этой проблеме заключается в использовании линейной регрессии, чтобы найти строку, которая наилучшим образом представляет набор данных. Используйте строку, чтобы определить ранг каждого элемента, начиная с самой дальней точки (0,0). Это показано на рис. 13.3. Угол линии определяет, какой из этих двух признаков (оценка или расстояние) будет иметь наибольшее значение.

Рисунок 13.3 также позволяет лучше понять задачу. У вас есть два различных признака: расстояние и средняя оценка. Я получил рисунок, показанный ниже. Если изменить угол линии, кафе могут отсортироваться в другом порядке. Я надеюсь, что это помогло.

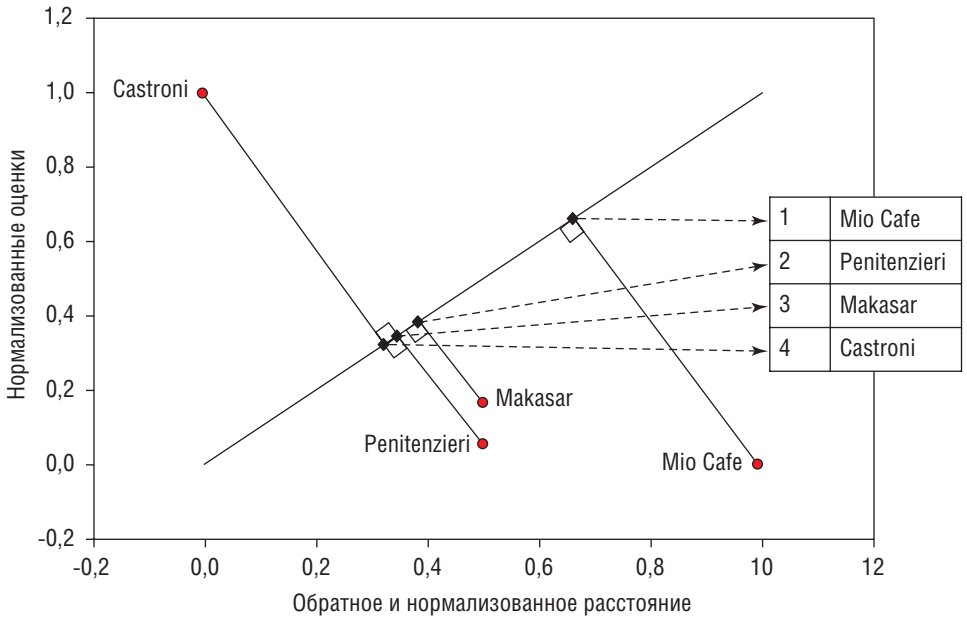


Рис. 13.3. Проецирование точки на линии показывает порядок элементов

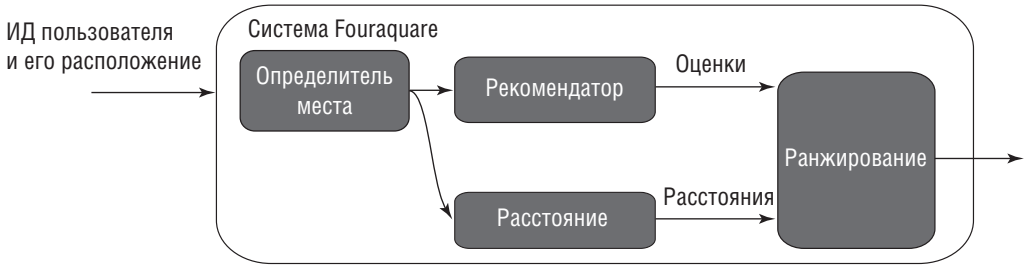


Рис. 13.4. Упрощенный вид системы ранжирования Foursquare

Вернемся к примеру Foursquare: если бы вы были Foursquare, вы бы, вероятно, сделали процедуру, показанную на рис. 13.4. Теперь появляется другая задача – как (представив, что вы – Foursquare) оптимизировать функцию, если вы не знаете, что она должна вернуть? Вы можете использовать признак посещения (почитайте статью). Задача определения того, как это должно выглядеть, не столь уж прямолинейна и проста.

13.2. Переранжирование

Если вы читали в предыдущей главе о гибридах, можете спросить, в чем разница между ранжированием и признако-взвешенным гибридом. Ответ: мы оптимизируем две разные вещи. Гибридный рекомендатор всегда прогнозирует оценки, а алгоритм LTR выполняет ранжирование. В следующем разделе мы рассмотрим, как определить эту задачу.

Некоторые пользователи назвали бы то, что делает Foursquare, переранжированием, поскольку оно формирует список мест, используя пространственный индекс, а это означает, что оно находит элементы ближе к вам, а затем сортирует список по оценке.

Простым примером переранжирования в рекомендательной системе будет использование сортировки по популярности, а затем переранжирование с помощью рекомендаторов. Популярность сужает список до наиболее популярных элементов и снижает риск отображения чересчур специфических (и, возможно, непопулярных). Это похоже на фильтрующий пузырь, но помните, что если пользователь предпочитает необычные элементы, то они будут двигаться вверх в списке.

В качестве примера посмотрите на Castroni (рис. 13.3). Эта кофейня далеко, но так как ее средняя оценка очень высока, она попадает в топ-4. Mio Caffé имеет оценки похуже, но она прямо рядом, так что, даже будучи непопулярной, она будет первой за счет близости.

Совместные алгоритмы фильтрации склонны рекомендовать элементы, понравившиеся нескольким людям, но при этом сильно понравившиеся. Алгоритм не имеет ни малейшего понятия о популярности и может быть использован для повторного ранжирования. Этот пример также описан на Netflix TechBlog[1].

[1] Подробнеее о TechBlog Netflix: **mng.bz/LacO**.

Вместо того чтобы повторно ранжировать элементы и прогнозы оценок, почему бы не начать с цели ранжирования и ориентировать алгоритм именно под эту цель? Алгоритмы LTR как раз это и делают.

13.3. Еще раз – что такое обучение ранжированию?

Рекомендатор или другой алгоритм, который создает ранжированные списки, обучается с помощью семейства алгоритмов *обучения ранжированию* (англ. *Learning to Rank* – LTR). У ранжирующего рекомендатора есть каталог элементов. Система извлекает элементы, которые имеют отношение к пользователю, а затем ранжирует их, выводя наиболее применимые.

Ранжирование осуществляется с использованием модели ранжирования[1]. Модель ранжирования обучается с использованием алгоритма LTR – это алгоритм обучения с учителем, а это означает, что вы должны предоставить ему входные и выходные данные. В вашем случае это `user_id` в качестве входных данных и ранжированный список элементов в качестве выхода. Это семейство алгоритмов имеет три подгруппы – точечные, парные и списочные. Кратко рассмотрим их все.

13.3.1. Три типа алгоритмов LTR

Алгоритмы LTR отличаются способами, которыми они выполняют оценку ранжированного списка во время обучения. На рис. 13.5 показаны все три типа.

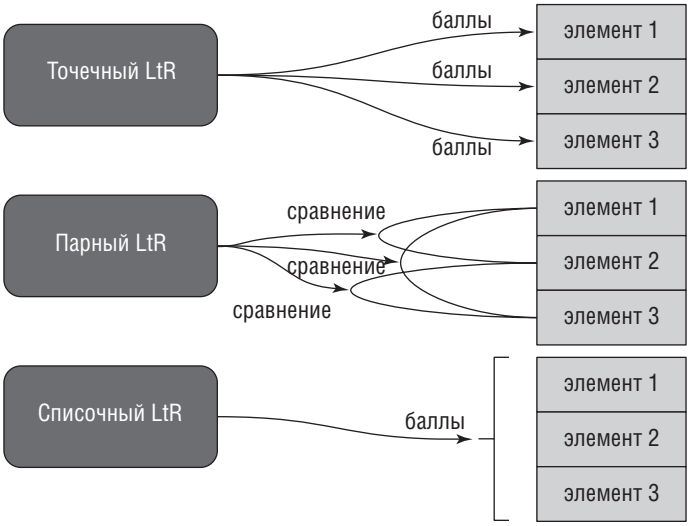


Рис. 13.5. Три типа алгоритмов LTR: точечные, парные и списочные

[1] Это определение взято из статьи «Краткое введение в обучение ранжированию» по Ханга Л.И. Ссылка: **times.cs.uiuc.edu/course/598f16/l2r.pdf**.

Точечные

Точечный алгоритм работает точно так же, как и рекомендаторы, рассмотренные в предыдущих главах. Он выполняет оценку для каждого элемента, а затем ранжирует их соответствующим образом. Отличие между прогнозированием и ранжированием заключается в том, что в случае с ранжированием не нужно думать о слишком больших значениях или масштабировании, так как балл обозначает всего лишь положение в списке.

Парные

Парные алгоритмы – разновидность бинарного классификатора. Это функция, которая принимает два элемента и возвращает их в определенном порядке. В парном ранжировании обычно оптимизируют выходные данные так, чтобы получить минимальное количество обменов элементов по сравнению с оптимальным ранжированием.

Для выполнения парного ранжирования вам нужно *абсолютное упорядочение*. Абсолютное упорядочение означает, что для любых двух элементов контента в каталоге можно сказать, какой из них более релевантен.

> **ПРИМЕЧАНИЕ.** Если вы реализовали парное ранжирование с использованием модели окрестности из главы 8, абсолютной упорядоченности достичь не получится, потому что алгоритм не может предсказать оценки для всех элементов.

Списочные

Это король алгоритмов LTR, потому что он рассматривает весь ранжированный список и оптимизирует. Преимущество списочного ранжирования заключается в том, что в нем правильная сортировка в верхней части списка считается важнее, чем в верхней. В предыдущих двух типах такой разницы нет.

Рассмотрим, к примеру, рекомендации топ-10. Парная рекомендация будет наказывать вас за неправильный порядок двух последних элементов, так же как для двух первых элементов. Это не здорово, ведь пользователям важнее верхняя часть списка. Но для реализации такого алгоритма нужно рассматривать весь список целиком, а не элементы попарно.

На словах звучит просто: вы должны ранжировать элементы таким образом, чтобы порядок элементов всегда был правильным. Но, оказывается, трудно программно вычислить, является ли один список лучше другого.

Чтобы посмотреть на алгоритм списочного ранжирования, я предлагаю CoFiRank (совместная фильтрация для ранжирования), который был представлен на NIPS (конференция по нейросетевой обработке информации) в 2007 году[1]. В следующем разделе мы рассмотрим алгоритм, где реализовано парное ранжирование.

---

[1] Веймер и соавт. Максимум. «Совместная фильтрация для ранжирования CoFiRank». Ссылка: **mng.bz/m0t1**.

13.4. Байесовское персонализированное ранжирование

Опять же, всегда стоит убедиться, что вы и ваши коллеги согласны с постановкой задачи. Это и в целом в жизни верно. Давайте начнем с определения проблемы или задачи, которую нужно решить. Чтобы решить эту задачу, мы будем использовать алгоритм, называемый байесовским персонализированным ранжированием (BPR), который был представлен в работе Штеффена Ренле[1].

Задача

Общая идея заключается в том, что нам нужно предоставить клиентам список элементов, где верхний будет самым релевантным и далее – по убыванию. Я уверен, что мы поняли друг друга. Теперь нужно описать задачу таким образом, чтобы и вы, и машина поняли это одинаково.

Нужно определить сортировку так: независимо от того, какие два элемента мы возьмем, один из них будет ранжирован лучше другого. Для того чтобы сделать это, нам нужны три правила: полнота, несимметрия и транзитивность. Для каждого данного пользователя сортировка будет определена следующим образом:

- *полнота* – для всех i, j в I (все элементы), если i не равно j, имеем либо $i >_u j$, либо $j >_u i$;
- *несимметрия* – для всех i, j в I, если $i >_u j$ и $j >_u i$, то $i = j$;
- *транзитивность* – для всех i, j, k в I, если $i >_u j$ и $j >_u k$, то $i >_u k$.

Казалось бы, мы зря тратим на это время, но это необходимо, чтобы заставить BPR работать.

Если у нас неявные данные

Алгоритм, о котором мы говорим, часто используется только при наличии неявной обратной связи. Но здесь проблема в том, что у нас не бывает отрицательной обратной связи, а есть только события, например совершение покупки. Отсутствие отрицательных данных делает трудным для алгоритма машинного обучения понять, когда он делает что-то не так, ведь он не знает, что такое хорошо и что такое плохо.

Отсутствие события может означать, что пользователь не знает, существует ли элемент вообще (например, не видел его). А может, он видел элемент, но ему не понравилось. А может, видел, понравилось, но пока не успел купить. В любом случае можно предположить, что отсутствие события хуже, чем его наличие, как показано на рис. 13.6. Независимо от того, купил пользователь товар или нет (квадратики на схеме). Теперь у вас есть разные условия для отношений между пользователями и элементами.

---

[1] Штеффен Рендл и др «БПР: Байесовское персонализированное ранжирование для неявной обратной связи». **arxiv.org/pdf/ 1205.2618.pdf**.

Если у вас есть два элемента – купленный и не купленный, – тогда при выполнении ранжирования вы можете определить, что купленный товар всегда является более привлекательным, чем тот, который не купили. Определившись с этим, вы можете рассмотреть пары элементов, каждый из которых может быть куплен или не куплен, но о них пока сложно сказать что-то.

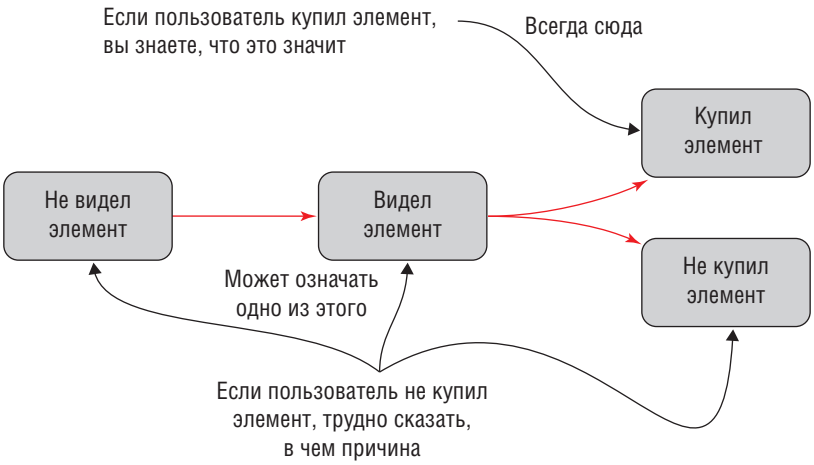


Рис. 13.6. Различные состояния отношений пользователь–элемент. С покупкой все понятно, а вот если пользователь не купил элемент – в чем причина?

На рис. 13.7 видно преобразование журнала покупки пользователя в матрицу заказа. На рис. 13.8 показано, как это расширяет ваши данные, – у каждого пользователя будет его собственная матрица.

Это все хорошо, но у вас есть данные – как использовать их здесь? Если вы хотите получить более точную выборку, посмотрите на алгоритм MF-BPR (байесовское персонализированное ранжирование с несколькими обратными связями). Послушайте лекцию с RecSys 2016 года на YouTube: «Bayesian Personalized Ranking with Multi-Channel User Feedback»[1].

При наличии неявных данных

Если у вас есть явные данные – оценки, – то вы могли бы сказать, что элементы без оценок имеют ранг ниже, чем элементы с оценками (положим, что оценка = покупка). Можно спросить, что делать с ними – ставить им среднюю оценку или оценку ниже всех остальных. На рис. 13.8 мы предполагаем, что оценка соответствует покупке.

Обучающие данные

Подход, описанный в предыдущем разделе, позволяет получить набор данных, который будет использоваться для подготовки ранжирующего рекомендатора. Этот набор данных будет содержать все кортежи (u, i, j), где i было куплено и оценено, а j – нет.

[1] Ссылка на видео: **www.youtube.com/watch?v=aKHLf4P3N08**.

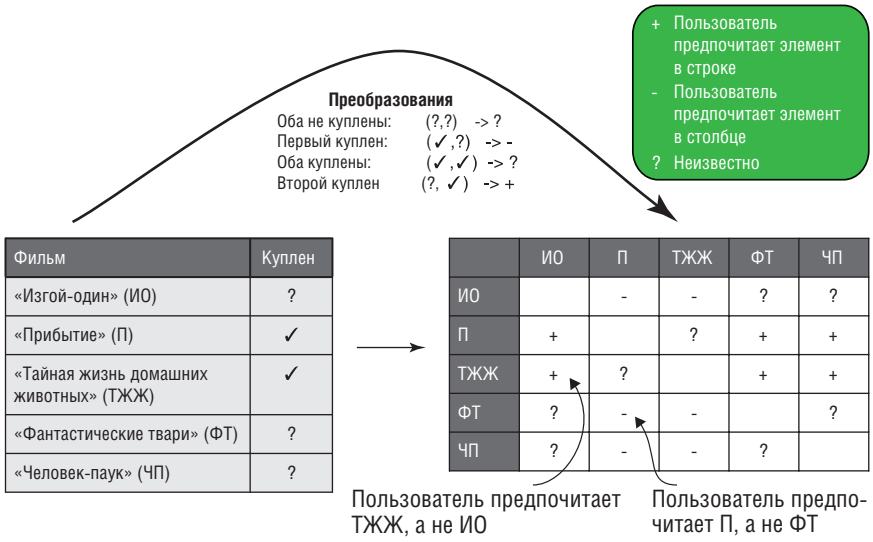


Рис. 13.7. Как преобразовать покупку пользователя в матрицу. Галочка означает, что пользователь купил элемент. В матрице справа «+» означает, что пользователь предпочитает элемент в в строке, а «–» означает, что пользователь предпочитает элемент в столбце

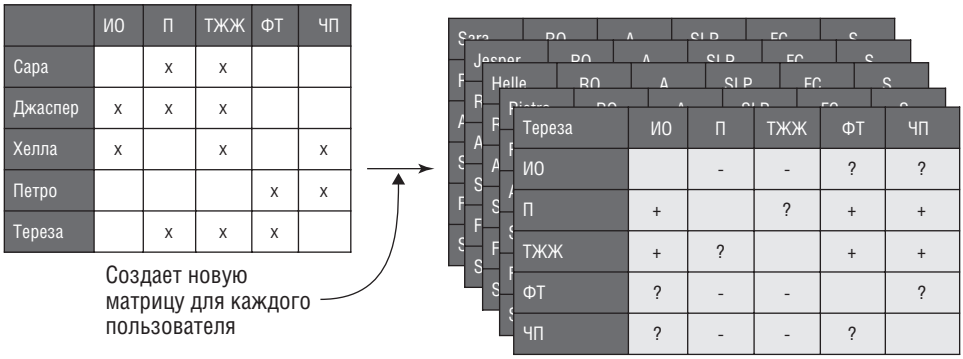


Рис. 13.8. Алгоритм создает матрицу порядка для каждого пользователя. Пять пользователей находятся в рейтинговой матрице, и будет генерировать пять матриц

13.4.1. Ранжирование с BPR

Нам нужно вычислить персонализированное ранжирование для всех элементов и всех пользователей в базе данных. Для персонализированного ранжирования будем использовать так называемую байесовскую статистику. Она строится на основе этого простого уравнения:

$$p(A|B) = \frac{p(B|A)p(A)}{p(B)}.$$

Это уравнение показывает, что вероятность (p) события A при условии, что B произошло, равна вероятности события A, умноженной на вероятность со-

бытия *B*, если *A* произошло, деленное на вероятность *B*. Пусть теперь *A* – был дождь, а *B* –дорога намокнет. Тогда Байес говорит, что вероятность того, что прошел дождь, учитывая, что на улице мокро, ($p(A|B)$), равна вероятности того, что шел дождь ($p(A)$), умноженная на вероятность того, что на улице влажно, если прошел дождь ($p(B|A)$), и деленная на вероятность того, что на улице влажно ($p(B)$). Это простое уравнение превратилось в интересное ответвление статистики, поэтому рекомендую посмотреть его.

Задачу ранжирования можно сформулировать так: имеем неизвестный порядок для каждого пользователя, который обозначаем как $>_u$ для пользователя *u*. $>_u$ – это полная упорядоченность, показывающая, что из любых двух элементов контента пользователь предпочтет один или другой. Мы также говорим, что θ – это список параметров рекомендательной системы, которые необходимо найти (как и для любой модели машинного обучения). Если речь идет о Funk SVD, помните, что задача сводится к тому, чтобы создать две матрицы, которые можно использовать для расчета прогнозов, как показано далее:

$$\begin{bmatrix} 5 & 3 & 0 & 2 & 2 & 2 \\ 4 & 3 & 4 & 0 & 3 & 3 \\ 5 & 2 & 5 & 2 & 1 & 1 \\ 3 & 5 & 3 & 0 & 1 & 1 \\ 3 & 3 & 3 & 2 & 4 & 5 \\ 2 & 3 & 2 & 3 & 5 & 5 \end{bmatrix} = \begin{bmatrix} u_{1,1} & u_{1,2} \\ u_{2,1} & u_{2,2} \\ u_{3,1} & u_{3,2} \\ u_{4,1} & u_{4,2} \\ u_{5,1} & u_{5,2} \\ u_{6,1} & u_{6,2} \end{bmatrix} \begin{bmatrix} v_{1,1} & v_{1,2} & v_{1,3} & v_{1,4} & v_{1,5} & v_{1,6} \\ v_{2,1} & v_{2,2} & v_{2,3} & v_{2,4} & v_{2,5} & v_{2,6} \end{bmatrix}.$$

Когда мы говорим о θ в этом контексте, θ – это множество всех $u_{i,j}$ и $v_{i,j}$. В BPR нужно найти такую модель, чтобы получилась самая высокая вероятность того, что эта модель выстроит идеальный порядок для всех пользователей. Вероятность может быть записана следующим образом:

$$p\left(\theta \middle| >_u\right)$$

(это читается так: вероятность увидеть θ (тета), учитывая порядок $>_u$). Из теоремы Байеса вы знаете, что если нужно максимизировать это, то это то же самое, что максимизировать следующее, так как они пропорциональны друг другу[1]:

$$p\left(>_u \middle| \theta\right) p(\theta).$$

Обратите внимание на то, что порядок $>_u$ и модель θ поменялись местами. Теперь вы будете работать с вероятностью увидеть порядок $>_u$ при имеющейся модели θ и умножать это на вероятность увидеть эту модель.

> **ПРИМЕЧАНИЕ.** В следующем разделе будет немного магии математики. Вы можете пропустить его, если мелкие детали вам не интересны.

Но, прежде чем перейти к магии, давайте чуть остановимся на том, что происходит. Мы предполагаем, что в идеальном мире существует способ одно-

[1] Подробнее: **en.wikipedia.org/wiki/Bayes'_theorem**.

значно упорядочить все элементы контента для каждого из пользователей, а это как раз и есть порядок $>_u$. Если существует такой порядок, есть также вероятность того, что существует алгоритм, который сможет сгенерировать его.

$p(\theta|>_u)$ – вот, что нам нужно. Если такой порядок существует, то какова вероятность того, что вы найдете модель, которая его создаст? Добавим сюда Байеса и спросим по-другому. Положим, что $p(\theta|>_u)$ – это то же самое, что спросить: какова вероятность того, что существует такое произведение θ и вероятности, что при имеющемся θ мы получим порядок? Стало яснее?

13.4.2. Магия математики (продвинутое колдовство)

Давайте посмотрим на две части выражения из предыдущего раздела. Для освежения памяти:

$$p(>_u|\theta)p(\theta).$$

Предположим, что мы имеем нормальное распределение

Давайте начнем с последней части уравнения: ($p(\theta)$). Предположим, что параметры модели являются независимыми и каждый из них имеет нормальное распределение ($p(\theta) \sim N(0,\Sigma_\theta)$ с нулевым матожиданием и ковариационной матрицей Σ_θ[1]. **Тогда последнюю часть можно переписать как:**

$$p(\theta) = \sqrt{\frac{1}{2\pi}} e^{-\frac{1}{2}\lambda\theta^2}.$$

Тогда $\Sigma_\theta = \lambda_\theta I$.

Функция правдоподобия

Переходя к $p(>_u|\theta)$, вы можете кое-что переписать. Когда вы говорите $>_w$, это касается только одного пользователя, но нам нужно максимизировать его для всех пользователей, т. е. максимизировать:

$$p(>_{u_1}|\theta) * p(>_{u_2}|\theta) * \ldots * p(>_{u_n}|\theta).$$

Это уравнение можно записать более компактно: $\Pi_{u \in U} p(>_u|\theta)$. У вас есть произведение вероятности того, что существует порядок для каждого пользователя при имеющейся модели. Вероятность упорядочения для одного пользователя должна быть такой же, как вероятность для всех пар элементов, где один был куплен, а второй – нет.

Ранее мы сказали, что рассматриваем только эти случаи. Используя здравый смысл и несколько хитрых трюков, вы можете сократить предыдущее произведение до произведения, содержащего вероятность упорядочения для каждой точки данных (u, i, j), т. е. все ваши данные D превращаются в

[1] Подробнее: **en.wikipedia.org/wiki/Normal_distribution**.

$$\prod_{u \in U} p(>_u | \theta) = \prod_{(u,i,j) \in D_s} p(i >_u j | \theta).$$

Вы можете сделать еще один шаг вперед. Поскольку θ – это модель рекомендательной системы, вы знаете, что $p(i >_u j | \theta)$ означает, что вы ищете вероятность того, что существует рекомендатор, который позволит спрогнозировать оценки таким образом, что $r_{ui} - r_{uj} > 0$. Еще раз перепишем произведение:

$$\prod_{(u,i,j) \in D_s} p(i >_u j | \theta) = \prod_{(u,i,j) \in D_s} p(r_{ui} - r_{uj} > 0).$$

Упрощаем порядок

Ранее я сказал, что проблема с ранжированием была бинарной, в том смысле, что у элемента *i* было четкое предпочтение. Поскольку у вас есть полное упорядочение, это описывает так называемую *функцию Хевисайда* (рис. 13.9). Ответом на вопрос $i >_u j$ может быть только {да, нет} (да или нет), учитывая определенную модель. Это означает, что $p(i >_u j | \theta)$ равна 0 или 1.

Мы смотрим только на данные, где один элемент был куплен пользователем, а другой не был. Это означает, что скольжения по функции у нас нет. Сначала она равна 0, а потом резко становится равна 1:

$$p(i >_u j | \theta) = \begin{cases} 1 \text{ если } i >_u j \\ 0 \text{ в противном случае} \end{cases}.$$

Помните, что в главе 11 мы говорили об оптимизации аналога, и это не работает, когда фукнция в один момент равна 1, а в другой – 0. С функцией Хевисайда трудно сказать, в каком направлении идти вниз безопасно. Чтобы решить эту проблему, вы можете использовать другую функцию, которая почти такая же – сигмовидная функция. Эта функция также работает в интервале от 0 до 1 и движется почти так же, как функция Хевисайда. Сигмоида определяются следующим образом:

$$\delta(x) = \frac{1}{1 + e^{-(x)}}.$$

На рис. 13.9 показана сигмоида в действии. Как видно из рисунка, вы можете вставить сигмоиду, не теряя слишком много целостности. Получим, что

$$p(i >_u j | \theta) = p(r_{ui} - r_{uj} > 0) = \delta(r_{ui} - r_{uj}) = \frac{1}{1 + e^{-(r_{ui} - r_{uj})}},$$

где $r_{ui} - r_{uj}$ – это прогноз оценки от рекомендатора. Теперь у нас есть все части – можно их совместить, придумать код на Python и выполнить ранжирование.

Опять же, нам нужно найти такой набор параметров для модели, чтобы получить максимальную вероятность выдать идеальное ранжирование для всех пользователей.

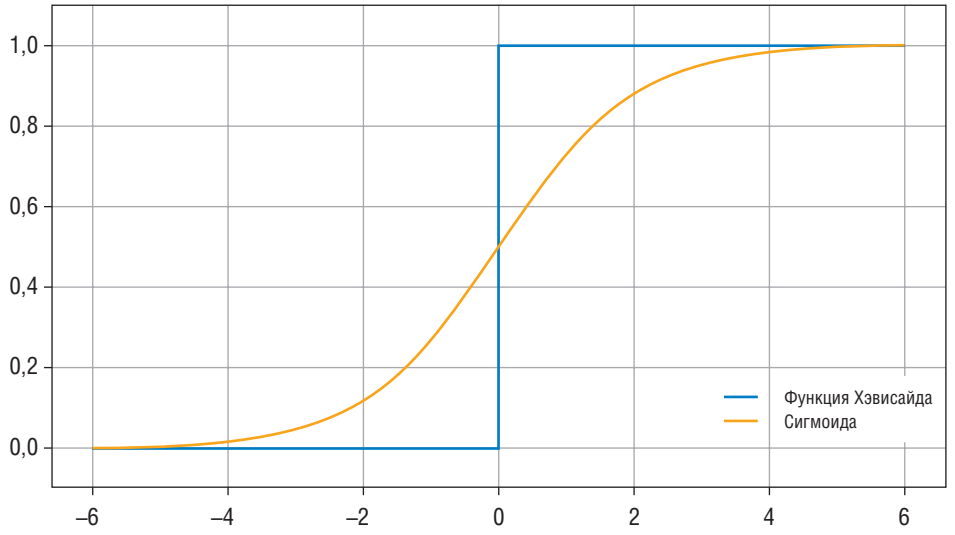


Рис. 13.9. Функция Хэвисайда и сигмоида в интервале от –6 до 6

Можно сказать, что нам нужно максимизировать следующее (под argmax имеются в виду параметры, которые максимизируют выражение):

$$\underset{\theta}{\operatorname{argmax}}\, p\left(\theta \mid >_u\right) p\theta.$$

Воспользуемся трюком, который позволяет привести нашу задачу к максимизации следующего, так как функция непрерывна и постоянно растет:

$$\underset{\theta}{\operatorname{argmax}} \ln\left(p\left(\theta \mid >_u\right) p\theta\right).$$

Тогда мы получим:

$$\underset{\theta}{\operatorname{argmax}} \ln\left(\prod_{(u,i,j)\in D_s} \delta\left(r_{ui} - r_{uj}\right) \times \sqrt{\frac{1}{2\pi}} e^{-\frac{1}{2}\lambda\theta^2}\right),$$

где D_s – это все комбинации, которые есть в ваших данных (пользователь u купил или оценил элементы i, но не элемент j).

Функция, которую вы добавили (ln), – это натуральный логарифм, и она имеет некоторые полезные свойства, показанные здесь[1]:

$$(\ln(a * b) = \ln(a) + \ln(b) \text{and} \left(e^{-\frac{1}{2}\lambda\theta^2}\right) = -\frac{1}{2}\lambda\theta^2).$$

Также положим, $\lambda := \frac{1}{2}\lambda$, что понадобится нам для упрощения выражения.

[1] Подробнее: **en.wikipedia.org/wiki/Natural_logarithm#Properties**.

Ну, хотя бы немного. Часть внутри argmax называется критерием оптимизации BPR (BPR-OPT):

$$\underset{\theta}{\operatorname{argmax}} \sum_{(u,i,j)\in D_s} \ln\left(\delta\left(r_{ui} - r_{uj}\right)\right) - \lambda \| \theta \|^2 .$$

Вот наша задача. Мы ничего не упустили? Давайте пройдемся еще разок.

Теперь наша проблема сводится к тому, чтобы определить рекомендательную систему. Запуск с набором параметров θ сделает все выражение очень большим, а это означает, что именно θ с самой высокой вероятностью даст то, что система создаст упорядочение $>_u$, которое соответствует предпочтениям всех пользователей. Это благородная цель, вам не кажется?

Но это была лишь постановка задачи. Теперь у нас много работы, ведь нужно создать алгоритм, который решает ее. Помните наш стохастический градиентный спуск из 11-й главы? Мы будем использовать что-то подобное и здесь.

Нам нужно найти градиент предыдущего выражения, чтобы понять, каким образом вы должны двигаться, чтобы приблизиться к оптимальному ранжированию. Отметим (без доказательства), что градиент BPR-OPT пропорционален следующему (значок бесконечности означает пропорциональность):

$$\frac{\delta \text{BPR} - \text{OPT}}{\delta\theta} \propto \sum_{(u,i,j) \in D_s} \frac{e^{-\left(r_{ui} - r_{uj}\right)}}{1 + e^{-\left(r_{ui} - r_{uj}\right)}} \times \frac{\delta}{\delta\theta}\left(r_{ui} - r_{uj}\right) - \lambda\theta.$$

И это функция, которую вы хотите использовать, чтобы выяснить, в каком направлении нужно двигаться, чтобы оптимизировать метод ранжирования. На этом закончим с магией математики и, притворившись, будто ничего странного не произошло, оптимизируем полученное выражение.

13.4.3. Алгоритм BPR

В статье, где описывается BPR, автор также предлагает алгоритм, называемый LearnBPR, который выглядит следующим образом:

1: **procedure** LEARNBPR(D_S, Θ)
2: initialize Θ
3: **repeat**
4: draw (u, i, j) from D_S
5: $\Theta \leftarrow \Theta + \alpha\left(\frac{e^{-\hat{x}_{uij}}}{1+e^{-\hat{x}_{uij}}} \cdot \frac{\partial}{\partial\Theta}\hat{x}_{uij} + \lambda_\Theta \cdot \Theta\right)$
6: **until** convergence
7: **return** $\hat{\Theta}$
8: **end procedure**

Это оно. Держу пари, вы чувствуете себя так, будто смотрели завернутый детектив, а затем проспали последние 10 мин (где как раз объяснялось, почему дворецкий сделал это). Но мы пришли к алгоритму, который будет производить ранжирование. До сих пор мы не говорили много об алгоритме рекомендатора, и, по сути, шаг

$$\theta \leftarrow \theta - \alpha \frac{\delta BPR - OPT}{\delta \theta}$$

в процедуре зависит от того, какой рекомендатор вы подключите к ней. В большинстве научных статей используется алгоритм матричной факторизации, так что мы с вами будем делать то же самое. Интересно, а если мы используем тот же алгоритм и эвристику, как в главе 11, получим ли мы такой же результат?

Не поверите, но у вас сейчас уже другая цель. Помните, что в главе 11 цель состояла в том, чтобы уменьшить разницу между оценками, которые у вас есть в вашей базе данных, и теми, которые предсказывает рекомендатор? Здесь не важно, какой тип оценки предсказывается, важен лишь порядок предсказания – ранжирование, которое позволяет обучению быть более «свободным». На данный момент также стоит отметить функцию вытяжки, которая может быть реализована несколькими различными способами и с различными стратегиями. Я использовал простейший вариант реализации, но есть и другие.

13.4.4. BPR с матричной факторизацией

Если вы забыли, что такое матричная факторизация, вы можете освежить вашу память, почитав главу 11. Здесь мы будем делать нечто подобное. Предсказание оценки в матричной факторизации сводится к умножению строки в матрице пользователя на столбец в матрице элементов **H**, что осуществляется с помощью следующего суммирования:

$$r_{u,j} = \sum_{f=1}^{K} w_{u,f} \times h_{j,f},$$

где K – количество скрытых факторов.

Чтобы подогнать это под выражение в примере, вы должны рассмотреть, как будет выглядеть градиент. Мы берем градиент относительно θ, что является объединением всех параметров, которые вы пытаетесь найти, т. е. всех w и h. Поразмыслив, вы увидите, что есть только три случая, которые нам интересны (из ненулевых):

$$\frac{\delta}{\delta\theta}\left(r_{u,i} - r_{u,j}\right)\begin{cases} \left(h_{u,i} - h_{u,j}\right) & if\ \theta = w_u \\ w_u & if\ \theta = h_i \\ -w_u & if\ \theta = h_j \\ 0 & else \end{cases}.$$

Возможно, нам нужно посмотреть на градиентное выражение, прежде чем понять это. Но я боюсь, что придется оставить это в качестве упражнения.

13.5. Реализация BPR

Алгоритм BPR впервые был описан в разделе 13.4. Авторы реализовали этот алгоритм библиотекой рекомендательной системы под названием MyMediaL-

ite в коде C#;, который вдохновлен этим[1]. На рис. 13.10 показано следующее завышенное представление о том, что вы реализуете в этом разделе:

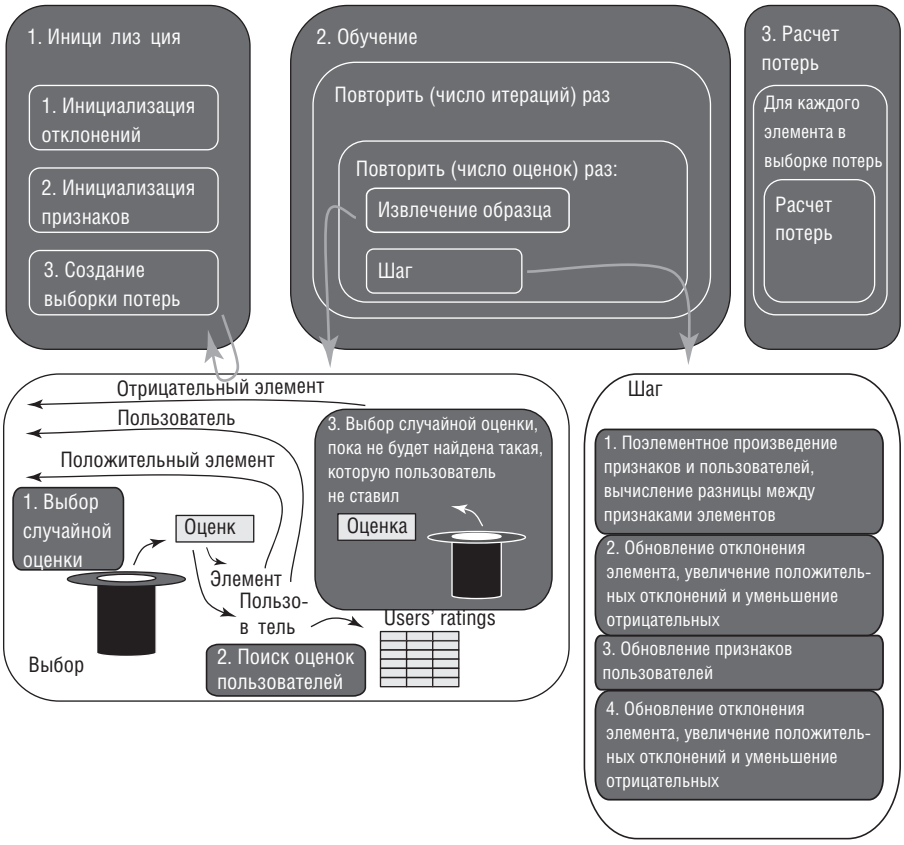


Рис. 13.10. Иллюстрация того, что мы собираемся реализовать в этом разделе. Мы начнем с инициализации, а затем с обучения. Для каждой итерации мы пройдем через одинаковое количество оценок и для каждого шага нарисуем образец пользователя, а также положительного и отрицательного элемента. Тогда мы делаем шаг, а это означает, что сдвигаем все факторы и отклонения в правильном направлении

Для запуска обучающего набора загрузите код из GitHub (**MNG.bz/04k5**) и следуйте инструкциям по установке в файле Readme. Затем перейдите в папку *MovieGEEKs* и выполните следующий листинг.

ЛИСТИНГ 13.1. Запуск обучающего алгоритма BPR

```
> python -m builder.bpr_calculator
```

Он выводит что-то вроде этого:

[1] Подробнее: **github.com/zenogantner/MyMediaLite**.

```
2017-11-19 16:23:59,147 : DEBUG : iteration 6 loss 2327.0428779398057
2017-11-19 16:24:01,776 : INFO : saving factors in ./models/bpr/2017-11-19
     16:22:04.441618//model/19/
```

Для того чтобы использовать эту модель, нужно имя папки, где была сохранена модель (факторы), и вставить ее в класс рекомендатора. Это может быть сделано автоматически в реальной системе. Но неплохо бы иметь ручной шаг, чтобы быть уверенным, что в производство не уйдет неисправная модель. Вставьте путь в файл *recs/bpr_recommender.py* в строку 17 или в качестве параметра по умолчанию для метода инициализации, как показано в следующем листинге. В аннотации указан относительный путь в журнале. Это результаты листинга 13.1? Если это так, укажите полный путь.

ЛИСТИНГ 13.2. Инициализация BPR

```
def __init__(self, save_path=''):
    self.save_path = save_path
    self.load_model(save_path)
    self.avg =
     list(Rating.objects.all().aggregate(Avg('rating')).values())[0]
```

Преобразование оценок в данные, пригодные для BPR

Прежде чем приступить к самому алгоритму, вы должны преобразовать данные оценок в пригодный вид. Алгоритм BPR использует неявную обратную связь, которая может быть реализована через щелчки или покупки. Если вы вспомните содержание главы 4, то можно сказать, что, если пользователь поставил чему-то оценку, значит, он это купил. Можно также сказать, что все оценки указывают на то, что пользователь что-то купил. Тогда возникает вопрос, хотите ли вы потерять информацию о том, что пользователь оценил элемент высоко.

Если вы хотите воспользоваться явной обратной связью, нужно превратить все оценки выше определенного порогового значения в факты покупки, а остальное удалить. Это уже вопрос интуиции. Здесь мы принимаем первое решение, так что у вас будет больше данных, которые можно использовать.

Метод LearnBPR

Во-первых, у вас есть общий метод сборки, который позволяет вам контролировать всю сборку. Метод сборки выглядит так, как показано в листинге. Вы можете просмотреть код для следующих записей в */build/bpr_calculator.py*.

ЛИСТИНГ 13.3. Общий метод сборки

```
def train(self, train_data, k=25, num_iterations=4):

    self.initialize_factors(train_data, k)   <-- Иници лиз ция ф кторов
```

```
    for iteration in range(num_iterations):    <-- Проход по num_iterators 4 р з

        for usr, pos, neg in self.draw(self.ratings.shape[0]):
            self.step(usr, pos, neg)    <-- Вызов метод ш г
```

Проход по всем сэмпл м, созд нным в методе `generate_samples`

Если вы ожидали чего-то крутого и яркого, то вы наверняка разочарованы, так что давайте двигаться дальше. Метод `initialize_factors` инициализирует все. Он не делает ничего сверхъестественного, так что я оставлю его вам – посмотрите, если интересно[1].

После того как метод инициализации запустится количество раз, указанное в параметре `num_iterations`, в каждой итерации осуществляется проход по *u*, *i*, *j*, т. е. пользователям, купившим элемент *i* и не купившим *j*.

В вашем случае выбор случайный. Для каждого вызывается метод шага (следующий листинг).

ЛИСТИНГ 13.4. Вызов метода шага

```
def step(self, u, i, j):

  lr = self.LearnRate
  ur = self.user_regularization
  br = self.bias_regularization

  pir = self.positive_item_regularization
  nir = self.negative_item_regularization

  ib = self.item_bias[i]
  jb = self.item_bias[j]

  u_dot_i = np.dot(self.user_factors[u, :],
                   self.item_factors[i, :] - self.item_factors[j, :])
  x = ib - jb + u_dot_i

  z = 1.0/(1.0 + exp(x))

  ib_update = z - br * ib
  self.item_bias[i] += lr * ib_update

  jb_update = - z - br * jb
  self.item_bias[j] += lr * jb_update
```

Созд ние коротких н зв ний для скорости обучения и конст нт регуляриз ции

Пок зыв ет то же с мое с отклонением элементов

Произведение между ф ктором пользов теля и р зличие между двумя вектор ми элементов

Обновление отклонений

[1] Подробнее: **mng.bz/tjAO**.

```
update_u = ((self.item_factors[i,:] - self.item_factors[j,:]) * z
             - ur * self.user_factors[u,:])
self.user_factors[u,:] += lr * update_u

update_i = (self.user_factors[u,:] * z
             - pir * self.item_factors[i,:])
self.item_factors[i,:] += lr * update_i

update_j = (-self.user_factors[u,:] * z
             - nir * self.item_factors[j,:])
self.item_factors[j,:] += lr * update_j
```

Обновление вектор ф кторов пользов теля

Обновление ф кторов элементов

Метод шага делает точно то же, что и метод матричной прогонки в главе 11. Я призываю вас прочитать его снова и вспомнить подробности (см. код в файле *bpr_calculator.py* и главе 11). В этой главе интереснее всего то, как берется образец и как выглядит функция прогнозирования и потерь.

Метод draw

Выборка состоит из идентификатора пользователя и двух идентификаторов элементов, где один элемент является более предпочтительным для пользователя по сравнению с другим. Это можно реализовать, сказав, что предпочтительным элементом является тот, который пользователь купил (или – в нашем случае – оценил). Для того чтобы получить такой образец из данных оценок, вы можете сделать следующее:

- возьмите случайную оценку пользователя, чтобы получить идентификатор пользователя и положительный элемент;
- продолжайте брать случайные оценки, пока у вас не останется элемент, который не оценен пользователем.

Это дает предположение о том, какие оценки нужно запомнить. Набор данных MovieTweetings содержит только тот контент, который кто-то оценил, так что в данных оценок возвращается весь контент. Кроме того, популярные элементы появляются чаще, чем другие, потому что они оценены больше.

В методе `draw` в листинге 13.5 используется `yield` вместо `return`, поэтому, когда дело доходит до `yield`, мы получаем результат. Но это остается в цикле `for`, так что `draw` будет перебирать все индексы. Вы можете сделать это, поместив все образцы в список, а затем вернув список. Но `yield` кажется более удачным способом сделать это. Обратите внимание, что сценарии следующих листингов можно найти в */build/bpr_calculator.py*.

ЛИСТИНГ 13.5. Метод draw

```
def draw(self, no=-1):
    if no == -1:
        no = self.ratings.nnz
```

```
    r_size = self.ratings.shape[0] - 1
    size = min(no, r_size)
    index_randomized = random.sample(range(0, r_size), size)
    for i in index_randomized:
        r = self.ratings[i]
        u = r[0]
        pos = r[1]

        user_items = self.ratings[self.ratings[:, 0] == u]
        neg = pos
        while neg in user_items:
            i2 = random.randint(0, r_size)
            r2 = self.ratings[i2]
            neg = r2[1]
        yield self.u_inx[u], self.i_inx[pos], self.i_inx[neg]
```

Поскольку данные нужно перемешать, мы создаем массив всех номеров индексов (0 в конце) и перемешиваем его

Проход по перемешанным индексам

Оценка выбрана, теперь нужно найти все оценки пользователя (здесь, возможно, стоит оптимизировать код, чтобы не нужно было каждый раз фильтровать все оценки)

Цикл идет до оценки отрицательного элемент

Хитрый трюк, чтобы снять ограничения цикл

Функция потерь (`create_loss_samples` в листинге 13.6) показывает, идете ли вы в правильном направлении. Она проходит через образец потери, который был создан в инициализации.

ЛИСТИНГ 13.6. Функция потерь

```
Build\bpr_calculator.py
def create_loss_samples(self):
    num_loss_samples = int(100 * len(self.user_ids) ** 0.5)
    self.loss_samples = [t for t in self.draw(num_loss_samples)]
```

Число образцов для отбор

Отбор образцов

Функция `loss`, показанная в листинге 13.7, проходит через образцы и рассчитывает

$$\sum_{(u,i,j)\varepsilon D_s} \ln\left(\delta\left(r_{ui} - r_{uj}\right)\right) - \lambda \| \theta \|.$$

ЛИСТИНГ 13.7. Расчет ошибки на данных потерь

```
def loss(self):
    br = self.bias_regularization
    ur = self.user_regularization
    pir = self.positive_item_regularization
    nir = self.negative_item_regularization

    ranking_loss = 0
    for u, i, j in self.loss_samples:
        x = self.predict(u, i) - self.predict(u, j)
        ranking_loss += 1.0 / (1.0 + exp(x))
```

Создает короткие названия констант

Расчет потерь ранжирования

```
Выр жения регуляриз ции
    c = 0
    for u, i, j in self.loss_samples:
        c += ur * np.dot(self.user_factors[u], self.user_factors[u])
        c += pir * np.dot(self.item_factors[i], self.item_factors[i])
        c += nir * np.dot(self.item_factors[j], self.item_factors[j])
        c += br * self.item_bias[i] ** 2
        c += br * self.item_bias[j] ** 2

     return ranking_loss + 0.5 * c
```

Потери р нжиров ния плюс половин регуляриз ции

В функции `loss` используется способ прогнозирования, но разница состоит в том, что предсказываются значения, а не оценки. В листинге 13.8 показано, как значения сравниваются с прогнозами другого элемента, и видно, как они должны быть ранжированы друг относительно друга.

ЛИСТИНГ 13.8. Ранжирование элементов относительно друг друга

```
def predict(self, user, item):
    i_fac = self.item_factors[item]
    u_fac = self.user_factors[user]

    pq = i_fac.dot(u_fac)

    return pq + self.item_bias[item]
```

Произведение между ф ктор ми элементов и пользов телей

Приб вление отклонений и возвр т

Выполнение этого алгоритма занимает много времени. Но в нем есть много мест, которые можно оптимизировать, тем самым ускорив его в пару сотен раз. В приведенном виде на моем MacBook 2017 года выполнение занимает около двух часов. При такой скорости выполнение 20 итераций займет 40 часов, так что вы можете сходить на пробежку или типа того. Однако после выполнения вы сможете использовать его, чтобы генерировать рекомендации. Давайте посмотрим, как это сделать.

13.5.1. Генерация рекомендаций

Чтобы проверить рекомендации, можете начать с MovieGEEKs, и, если модель уже обучена, она будет производить рекомендации BPR, используя метод, показанный в следующем листинге. Вы найдете код в файле в *recs/bpr_recommender.py*.

ЛИСТИНГ 13.9. Метод рекомендаций топ-*N* с помощью модели BPR

```
def recommend_items_by_ratings(self, user_id, active_user_items, num=6):

    rated_movies = {movie['movie_id']: movie['rating']
                    for movie in active_user_items}

    recs = {}
    if str(user_id) in self.user_factors.columns:

        user = self.user_factors[str(user_id)]

        scores = self.item_factors.T.dot(user)

        sorted_scores = scores.sort_values(ascending=False)
        result = sorted_scores[:num + len(rated_movies)]

        recs = {r[0]: {'prediction': r[1] + self.item_bias[r[0]]}
                for r in zip(result.index, result)
                if r[0] not in rated_movies}

    s_i = sorted(recs.items(),
                 key=lambda item: -float(item[1]['prediction']))

    return s_i[:num]
```

Создает словарь активных фильмов пользователя, что довольно удобно позволяет исключить рекомендации того, что пользователь уже видел

Убедитесь, что модель знакома с пользователем, иначе рекомендаций не получится

Рассчитайте произведения факторов элементов и пользователей для расчета сходств

Сортировка значений по убыванию

Обрезка списка до возвращаемого размера плюс количество оценок пользователя

Проход через результирующие элементы с применением отклонения

Сортировка и возврат ожидаемых значений

Чтобы сделать это, вы должны сначала загрузить модель с помощью кода, приведенного в листинге 13.10. Модель сохраняется на последнем шаге обучения.

ЛИСТИНГ 13.10. Загрузка модели

```
def load_model(self, save_path):

    with open(save_path + 'item_bias.data', 'rb') as ub_file:
        self.item_bias = pickle.load(ub_file)
    with open(save_path + 'user_factors.json', 'r') as infile:
        self.user_factors = pd.DataFrame(json.load(infile)).T
    with open(save_path + 'item_factors.json', 'r') as infile:
        self.item_factors = pd.DataFrame(json.load(infile)).T
```

13.6. Оценка

Как протестировать алгоритм? Один из способов – автономный тест, о котором мы говорили в главе 9, где использовали кросс-проверку. Процедура оценки показана на рис. 13.11.

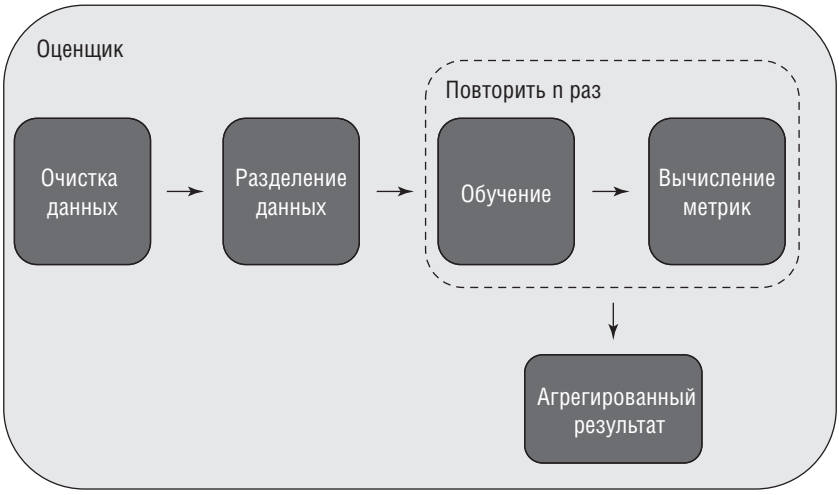


Рис. 13.11. Оценка алгоритма. В данной процедуре данные сначала очищаются, а затем разделяются на k частей для перекрестной проверки. Для каждой части выполняется обучение и оценка алгоритма. Когда это будет выполнено, вы агрегируете результат

В следующем листинге показан метод оценки, который я добавил. Это создает данные, которые показаны на графике на рис. 13.12. Вы можете просмотреть код для этого перечисления в файле */evaluator/evaluation_runner.py*.

ЛИСТИНГ 13.11. Метод оценки

```
def evaluate_bpr_recommender():
    timestr = time.strftime("%Y%m%d-%H%M%S")
    file_name = '{}-bpr-k.csv'.format(timestr)

    with open(file_name, 'a', 1) as logfile:
        logfile.write("rak,pak,mae,k,user_coverage,movie_coverage\n")

        for k in np.arange(10, 100, 10):
            recommender = BPRRecs()
            er = EvaluationRunner(0,
                                  None,
                                  recommender,
                                  k,
                                  params={'k': 10,
                                          'num_iterations': 20})
```

```
        result = er.calculate(1, 5)

        user_coverage, movie_coverage =
RecommenderCoverage(recommender).calculate_coverage()

        pak = result['pak']
        mae = result['mae']
        rak = result['rak']
```

Измерение точности дало мне результат, показанный на рис. 13.12. Впрочем, здесь ничего неожаднного. Но я уверен, что можно добиться лучшего: результат неплохой, но на малых k – не очень. Я также вычислил охват, и он тоже оказался лучше. Охват элементов составил 6,4 %, а охват пользователей – 99,9 %.

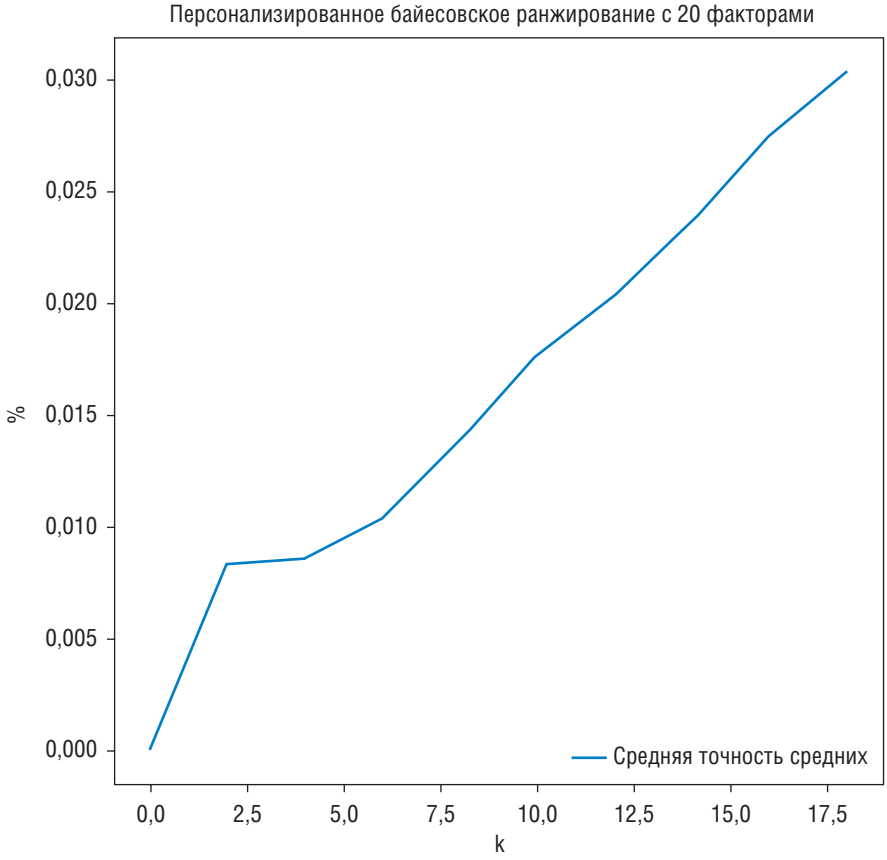


Рис. 13.12. Измерение средней точности для BHP на короткую верхнюю N. Это не так впечатляет, но я уверен, что он может быть изменен, чтобы сделать его лучше

Если проверить это на больших числах, то все окажется намного лучше, как показано на рис. 13.13. Но тут дело в том, что эти данные локальны для конкретного набора данных, и вы можете использовать их только в качестве

ориентира для дальнейшего усовершенствования. Это рекомендатор рекомендует только 6 % элементов, что не так уж много. Вероятно, стоит добавить сюда рекомендатор на основе контента или еще что-нибудь, чтобы у вас было больше рычагов и инструментов для настройки.

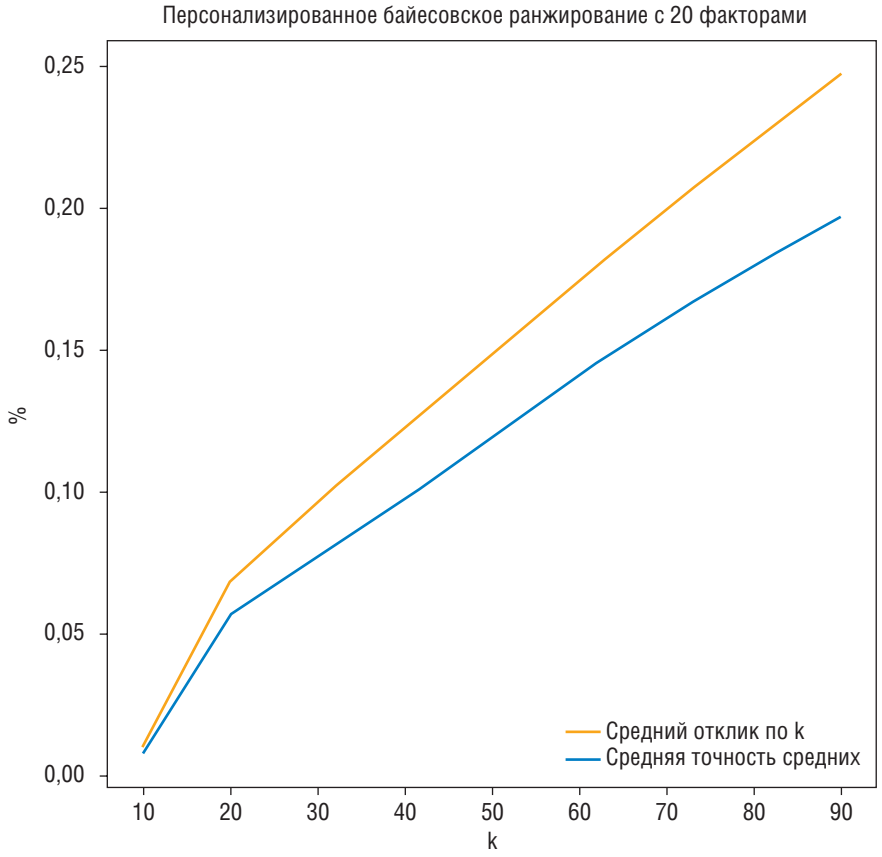


Рис. 13.13. Точность и отлик для алгоритма BPR

13.7. Эксперименты с BPR

- BPR представляет собой сложный алгоритм, и во время его реализации нужно принять множество решений. К сожалению, я пропустил многие из них, например сколько факторов следует включить и какая скорость обучения позволяет системе оптимизировать процесс? Это вовсе не означает, что это все неважные вещи, но их регулировка лежит уже на вас. Используйте все, что узнали в предыдущих главах, чтобы оценивать метапараметры. Давайте быстро пробежимся по ним.

У вас есть отклонение элемента и факторы пользователя, и нужно решить, сколько факторов использовать. Число факторов должно определяться тем, насколько сложен ваш набор данных. Например, если фильмы разделены на

небольшие наборы типов (или жанров), слишком большого количества факторов не требуется. Но если вы делаете рекомендатор вроде того, что есть на Vivino.com, то вам, вероятно, потребуется гораздо больше факторов. Это нужно проверять на каждом конкретном наборе данных.

Насчет скорости обучения трудно дать однозначные указания. Я обнаружил, что слишком высокая скорость обучения дала мне большие значения факторов пользователей, а значит, значения отклонений несут меньше информации, в то время как слишком низкая скорость обучения, наоборот, наделяет отклонения большей важностью. Регуляризация пытается уравновесить все это, но если значения факторов оказались великоваты, то скорость обучения надо уменьшать.

Я люблю сравнивать элементы, а затем выталкивать их в разных направлениях (речь идет именно об отклонении элементов). Это может быть скорректировано с помощью положительной и отрицательной регуляризации элементов. Также это влияет на то, как сильно выталкивается отрицательный элемент, и это может повредить новым элементам, если их посмотрело мало пользователей.

Набор данных, который вы использовали, содержит оценки, поэтому вы могли бы удалить элементы с низкими оценками, чтобы они не фигурировали в качестве положительных элементов в обучающих данных. Но, даже если фильм пользователю не понравился, его можно считать потребленным и извлечь из этой информации пользу.

Узнали ли вы в этой главе все, что вам нужно? Если нет, вы можете перечитать эту главу еще раз. Но это трудно, и, если вы не собираетесь реализовывать алгоритм BPR, сильно углубляться не стоит. Но если вы планируете реализовать рекомендатор с алгоритмом BPR, то нужно знать, как он работает. Обучение ранжированию – тоже важная вещь при работе с поисковыми машинами, так что узнать побольше о ранжировании тоже не помешает.

Вот и все. Осталась еще одна глава, и можно будет считать книгу прочитанной. Не забудьте оставить книге отзыв на GoodReads.com, неважно – хороший или плохой. Не забывайте, что рекомендательной системе нужна обратная связь!

Резюме

- Алгоритмы обучения ранжированию (LTR) решают задачи, отличные от классической задачи прогнозирования оценок.
- В Foursquare используется алгоритм ранжирования, объединяющий оценки и расположение мест, что является прекрасным примером того, как объединить релевантные данные в одну рекомендацию.
- Проблема алгоритма LTR заключается в том, что сложно придумать непрерывную функцию, которую можно оптимизировать.
- В этой главе мы коснулись теоремы Байеса, так что, даже если это не было предметом данной книги, ее стоит изучить, потому что она используется во многих сценариях. Рекомендую книгу «Practical Probablistic Programming» Ави Пфеффера (Manning, 2016).
- Байесовское персонализированное ранжирование (BPR) можно использовать поверх матричной факторизации, которую мы рассмотрели в главе 10, а также с другими типами алгоритмов.

Глава 14

Будущее рекомендательных систем

В этой главе мы отправимся назад в будущее:

- кратко пробежимся по содержанию книги;
- я приведу вам список тем, которые можно изучить, если вы хотите продолжить свое путешествие в захватывающий мир рекомендательных систем;
- несмотря на то что никто не знает, каково будущее рекомендательных систем, я сделаю пару предположений и выскажу кое-какие мысли.

Мне потребовалось три года, чтобы, наконец, начать писать эту главу. Я надеюсь, что ваш путь будет немного быстрее. Хотелось бы сказать, что теперь вы знаете все о рекомендательных системах, что вы теперь крутой эксперт, который знает все алгоритм рекомендаторов, и вас теперь ничем не удивить. Но это не так. Вы проделали долгий путь, но до становления экспертом еще далеко.

В этой книге вы узнали достаточно, чтобы погрузиться глубже в эту тему. Но не ограничивайтесь чтением. Поиграйте с новыми алгоритмами и ознакомьтесь поподробнее с теми, что описаны в книге. Вы найдете множество маленьких трюков, которые позволят сделать каждый из них лучше.

Будем надеяться, что сайт MovieGEEKs дал вам основы того, как загружать различные виды данных, и позволил вам попробовать некоторые из вещей, которые мы рассмотрели в этой книге. Помните, что сайт MovieGEEKs был реализован с целью понять рекомендательные системы и алгоритмы, и много чего на нем можно оптимизировать.

Но прежде чем отпустить вас в открытое плавание в мир рекомендательных систем, я хочу поговорить о нескольких вещах, которые не вместились в эту книгу. Мой редактор, вероятно, сочтет это темой для следующей книги, но моя жена против, так что вы, кажется, остаетесь сами по себе. Во-первых, давайте быстро пробежимся по темам, которые мы здесь рассмотрели.

14.1. Вся книга в паре предложений

В этой книге вы узнали о рекомендательных системах, которые в целом могут быть описаны схемой, показанной на рис. 14.1.

Мы начали с разговора о сборе данных. В известной работе Педро Доминго под названием «Несколько полезных вещей о машинной обучении» говорится, что чем больше данных, тем сложнее алгоритм[1].

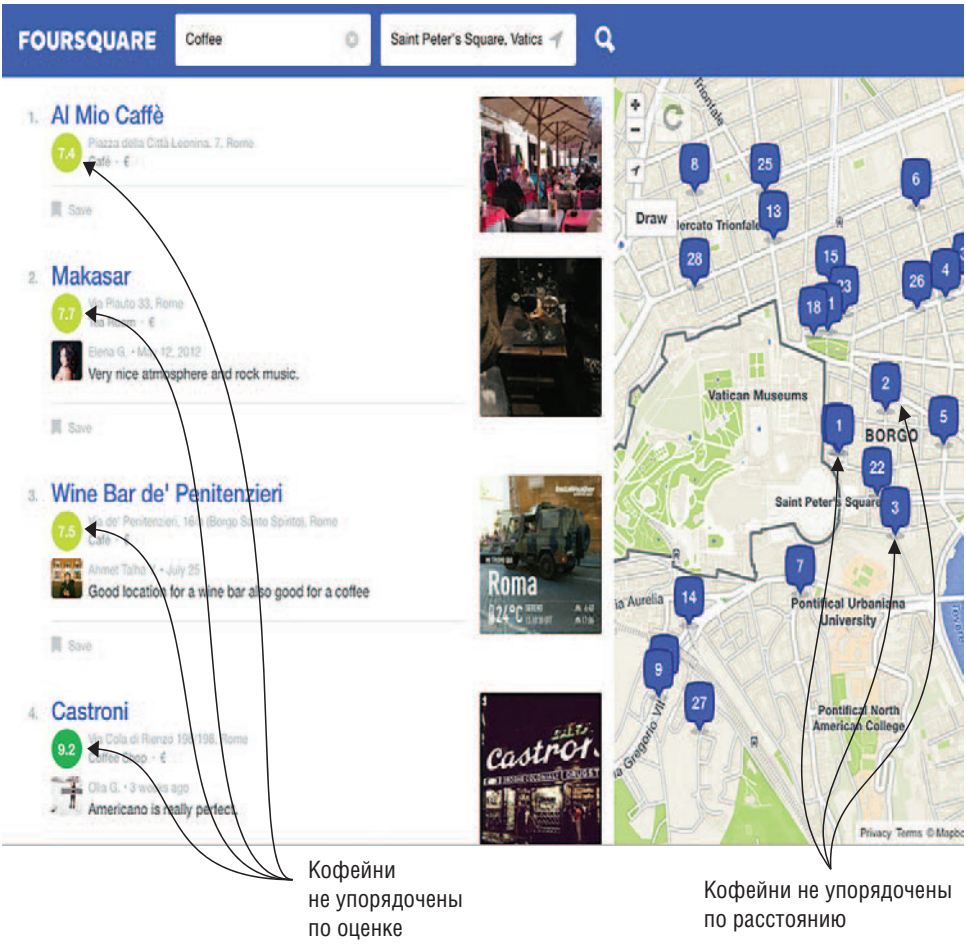


Рис. 14.1. Работа рекомендательной системы

Это также верно и для рекомендательных систем, но в них собранные данные не всегда являются источником истины.

Явные оценки могут быть отражением настроения или результатом социальных влияний и, таким образом, не всегда четко указывают на то, чего

[1] Подробнее: **homes.cs.washington.edu/~pedrod/papers/cacm12.pdf**.

хочет пользователь. *Неявные оценки*, как следует из названия, не поставлены пользователями напрямую, поэтому вы или машина должны расшифровать ту совокупность событий, которая лежит между каждым пользователем и элементом. Также не новость, что вкусы людей меняются с течением времени, так что старые данные могут быть неверными. Не всегда можно легко понять, какую информацию несут поведенческие данные. Превращение событий в оценки и кластеризация пользователей является частью подготовки данных для создания рекомендательной модели. Это часто называют предварительной обработкой данных.

В то время как данные важны для понимания того, чего хочет пользователь, не менее важно иметь возможность видеть, как работает система, поэтому нужны данные аналитики, которые будут держать вас в курсе ситуации.

При расчете рекомендаций часто используют модель, поэтому мы рассмотрели *обучение модели*. Вы можете разделить различные виды рекомендаций по уровням персонализации – от полностью неперсонализированных до очень персонализированных, как показано на рис. 14.2. Мы начали с неперсонализированных рекомендаций, поскольку в этом случае не требуется знать что-либо о пользователе. Это позволяет реализовать мягкий запуск при добавлении рекомендаций на ваш сайт.

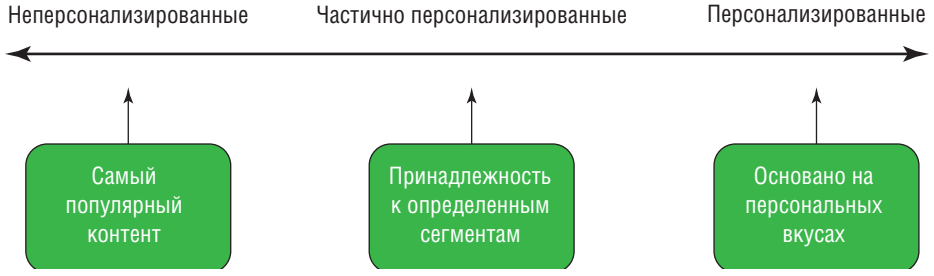


Рис. 14.2. Степени персонализации

Сперва мы рассмотрели неперсонализированные рекомендации, которые помогли найти самые продаваемые элементы, популярные элементы или самые высокооцененные элементы. Все пользователи начинают из «холодного» состояния, поэтому мы посвятили целую главу (глава 6) одной из наиболее сложных проблем в рекомендательных системах. Она заключается в том, чтобы выяснить, что делать с новыми пользователями и элементами. Попытка решить проблему рекомендаций для новых пользователей сводится к анализу корзины покупок.

Затем мы переключились на то, как сегментировать элементы (глава 7) с целью составлять полуперсонализированные рекомендации, учитывая данные пользователей.

Перед переходом к полной персонализации мы измерили расстояние и сходство между элементами и между пользователями, что нужно делать почти для каждого алгоритма рекомендательной системы.

Первый персонализированный метод рекомендаций – это *совместная фильтрация* (глава 8), которая дает рекомендации, основанные на сходстве между пользователями или элементами. Это может быть сделано с использованием оценок, явно поставленных пользователем или неявно путем анализа данных в журналах.

Есть два типа совместной фильтрации. Мы начали с *фильтрации в окрестности*, в которой алгоритм использует меры сходства и находит элементы или пользователей, близких к текущему. Ознакомившись с одним алгоритмом, мы изучили оценку алгоритмов (в главе 9). Рассмотрев множество различных метрик оценки рекомендатора, мы реализовали оценку через среднюю точность средних (MAP).

Если у вас нет данных о пользователе (или имеется подробное описание контента), то стоит рассмотреть рекомендатор на основе контента. У вас есть несколько способов сделать это, но суть в том, что мы смотрим на контент и рассчитываем сходство по его данным. Мы рассмотрели создание векторов, используя метод *частоты и обратной частоты документа* (TF-IDF), а затем с помощью *скрытого распределения Дирихле* (LDA) (в главе 10).

Изучив LDA, мы вернулись к совместной фильтрации, только теперь перешли к *фильтрации на основе модели*. Фильтрацию на основе модели можно сделать многими способами, главный из которых – *матричная факторизация* (глава 11). Здесь же мы внимательнее взглянули на алгоритмы машинного обучения. Поговорили о традиционном *сингулярном разложении* (SVD), а затем перешли к методу Funk SVD, который получил отличные результаты в конкурсе Netflix Prize.

Имея неплохой набор различных рекомендательных алгоритмов, мы начали думать, как их объединить тем или иным способом. Вы исследовали несколько таких способов в главе о гибридах, где мы реализовали *функционально-взвешенное линейное сочетание* (FWLS) – именно этот метод выиграл приз Netflix. Затем мы изучили новый тип алгоритма под названием *обучение ранжированию* (LTR), в котором упор идет не на правильное прогнозирование оценок, а на создание списков элементов, которые соответственно ранжируются. Эти алгоритмы могут быть разделены на три различных типа, как показано на рис. 14.3.

Каждый метод ранжирования имеет свои преимущества и недостатки. Я выбрал тот, о котором часто говорят, когда речь идет об обучении ранжированию, – *байесовский персонализированный алгоритм ранжирования* (BPR).

На этом этапе вы можете спросить – какой алгоритм все-таки использовать? Это зависит от того, какие у вас есть данные. Я предлагаю использовать матричную факторизацию. Когда она будет реализована, вы можете построить модель LTR поверх нее. Или можете добавить еще один алгоритм и создать ансамбль.

И наконец, пора нам отправиться в будущее и рассмотреть парочку перспективных рекомендаторов.

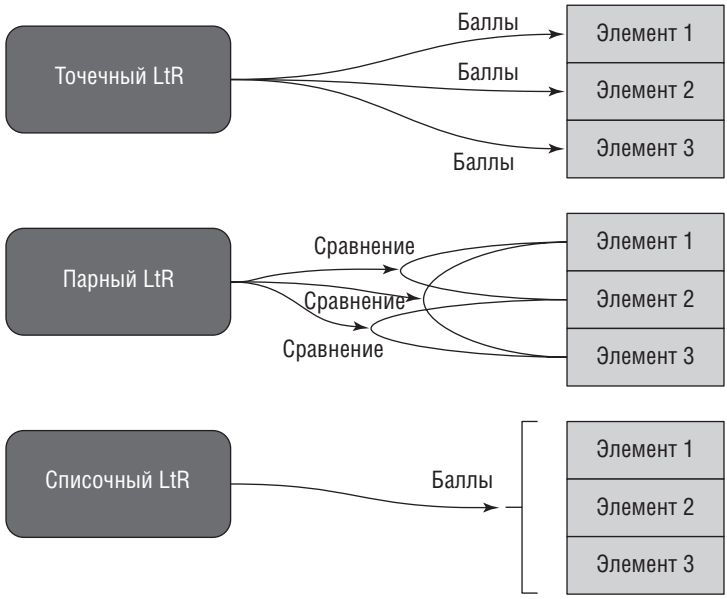


Рис. 14.3. Различные типы алгоритмов LTR

14.2. Темы для дальнейшего изучения

Вот что я рекомендую в качестве следующих шагов вашего освоения рекомендательных систем.

14.2.1. Дальнейшее чтение

Во-первых, если вы хотите больше подробностей и больше способов генерировать рекомендации, существует много исследований, которые можно рассмотреть. Я от всей души рекомендую книгу «Recommender Systems Handbook» Франческо Риччи и Лиора Рокач. (Springer, 2015). Это толстенный талмуд, но в нем рассказано больше, чем мне удалось вписать в эту книгу.

Я надеялся в этой книге поговорить побольше об онлайн-тестировании. Мы немного поговорили об A/B-тестировании в главе 9, а далее вам нужно почитать побольше об использовании/исследовании и многоруких бандитах. Для этой цели я рекомендую «Statistical Methods for Recommender System» Дипака Агарвала и Би-Чанг Чена из LinkedIn (Cambridge University Press, 2016).

Помимо книг, загляните на GroupLens (**grouplens.org**) и найдите материалы конференции ACM RecSys. Там было много интересных работ, и на YouTube есть их канал, посвященный рекомендательным системам (**mng.bz/ta38**).

Читая эту книгу, вы могли подумать, что совместная фильтрация, описанная в главе 8, – это устаревший подход, но исследования по нему все еще ведутся, и многие компании используют его. Например, работа года на RecSys2016 на-

зывалась «Локальные модели элемент–элемент для топ-*N*-рекомендаций»[1]. Вы можете посмотреть видео автора работы Еванджелины Кристакополоу на YoutTube.

Рекомендательные алгоритмы являются отраслью машинного обучения. Мало кто работает исключительно с рекомендательными алгоритмами. Я полагаю, что на вашем пути становления экспертом в этой области вам стоит изучить тему машинного обучения в целом. Рекомендую две книги: «Real-World Machine Learning» Хенрика Бринка и др. (Manning, 2016) и «Algorithms of the Intelligent Web», 2-е изд., Дуглас Г. МакЛиверейт и др. (Manning, 2016).

Поисковые системы тоже тесно связаны с рекомендациями. Я бы даже сказал, что поисковые системы являются частным случаем разрозненных рекомендательных систем. А эксперты в поисковых системах, например Дуги Тернбулл, сказали бы, что рекомендаторы являются поисковыми машинами. Таким образом, мы приходим к дискуссии о курице и яйце. В большинстве мест, где реализуются рекомендаторы, есть также поисковый индекс, поэтому обратите внимание на «Relevant Search» Дага Тернбулла и др. (Manning, 2016).

14.2.2. Алгоритмы

Обратимся теперь к алгоритмам Обучения ранжированию (LTR). Довольно популярными являются алгоритмы списочного типа, которые посложнее тех, что описывались в главе 13.

В настоящее время каждый серьезный специалист по машинному обучению в любой сфере обращается к глубокому обучению, и в сфере рекомендательных систем также ведутся активные исследования. Дело движется так быстро, что я даже не буду предлагать книги, потому что они устареют еще раньше, чем эта книга выйдет со станка. Тем не менее обратите внимание на конференцию Deep Learning Workshop в 2016 году (**dlrs-workshop.org/dlrs-2016/program/**).

Одна из проблем алгоритмов, описанных в этой книге, и одна из нерешенных проблем рекомендательных систем заключается в том, что большинство рекомендаторов оптимизированы на показ элементов, максимально похожих на то, что пользователь уже видел. Но нам нужен рекомендатор, который будет выдавать что-то новенькое и неизведанное. Это зависит от вашей конкретной задачи и целей, но стоит подумать о том, чтобы добавить немного хаоса в рекомендации. Еще одна проблема заключается в том, что вряд ли пользователю хочется видеть один и тот же список топ-*N*-рекомендаций снова и снова.

14.2.3. Контекст

Рекомендательные системы движутся от стационарных (в настольных версиях) к переносным (на всех видах мобильных устройств).

Пользователь может зайти на сайт из Европы, а потом из США. Поскольку устройство является мобильным, погода и другие условия тоже начинают иметь значение.

---

[1] Подробнее: **mng.bz/5c7E**.

Я обсуждал эту идею на протяжении всей книги, но не показывал каких-либо конкретных решений. В зависимости от вашей задачи, возможно, стоит также учитывать контекст. Например, в главе 11 мы узнали об алгоритме Funk SVD, который может также обрабатывать контекст, или можно использовать метод повторного ранжирования, описанный в главе 13.

14.2.4. Взаимодействие «Человек–Машина»

Я повернут на дата-сайенс и машинном обучении, поэтому вы, вероятно, хотели бы не думать о фронтенде и заниматься только рекомендатором. Но интерфейс – это тоже важно[1].

14.2.5. Выбор подходящей архитектуры

Давайте смотреть правде в глаза. Сайт MovieGEEKs работает так себе. Для одного конкретного пользователя все нормально, но я боюсь, что на большее он и не способен! Хорошо бы выбрать более мощную платформу для запуска рекомендатора и веб-сайта.

Django имеет неплохую производительность, но ей нужна реальная база данных. Я использовал SQLite, потому что ей нужно меньше настроек, но я рекомендую обновиться до PostgreSQL или аналогичной базы данных. Однако это не значит, что надо все бросить и перекидывать все на новую архитектуру и покупать новое оборудование. Во-первых, узнайте, даст ли рекомендатор вам то, что вы хотите. Если вы не можете протестировать его на нормальном трафике, это не имеет значения: протестируйте все на 1 % ваших клиентов и посмотрите, как они отреагируют. Только помните, что чем больше данных вы даете рекомендатору, тем лучше он работает (это общее правило).

Но давайте представим, что это вы уже сделали и хотите раскрутить свой рекомендатор до реальных оборотов. На этот случай у меня тоже есть парочка советов.

Облачное хранение не решит все ваши проблемы

Нас со всех сторон атакует реклама облачных сервисов, которые все один другого лучше, быстрее и дешевле. Но прежде чем перейти в облако, необходимо учесть следующее:

- *данные должны находиться в облаке не просто так*. Если у вас стационарная система (с локальными серверами), то вам придется как-то переместить все свои данные в облако, чтобы делать расчеты, необходимые для рекомендаций. Пропускная способность и вычислительные мощности могут выйти в копеечку;
- *готовые рекомендаторы «из коробки», такие как Microsoft Cognitive Services Recommendation API, вероятно, хорошо подходят для начала*. Но тут могут присутствовать ограничения на то, сколько элементов может быть

[1] Подробнее: **mng.bz/933q**. Другой вопрос в том, нужно ли рекомендовать всем одно и то же? У всех людей темперамент и настроение разные, так что рекомендатор должен это учитывать.

в каталоге и как часто можно к нему обращаться. Тщательно исследуйте ограничения и убедитесь, что можете позволить себе полагаться на стороннее решение;

- *всерьез подумайте о защите*. Если у вас есть какие-либо конфиденциальные данные, помните, что хранение в облаке обязывает вас соблюдать законы страны-сервера, которые применяются к данным, хранящимся в нем;
- *данные – наиболее ценный актив*. Не показывайте его другим, прежде чем подумайте, какие ценности вы предоставляете вашим конкурентам.

Из плюсов: облачные сервисы снимают немалую часть нагрузки с сервера и масштабируются в пиковые периоды. Я не говорю, что облако – это плохо, но следует крепко подумать перед его использованием.

Какую выбрать платформу для обработки

И наконец, вашему вниманию … ТА-ДАМ!… Вокруг платформы Spark столько хайпа, что я просто не мог не упомянуть ее хотя бы разочек.

Spark представляет собой распределенную вычислительную платформу, которая может реализовывать машинное обучение в распределенном формате. Если вы заинтересованы в выполнении вычислений на многих компьютерах, то Spark – ваш кандидат. Если вы посмотрите на Spark.MLlib (**spark.apache.org/mllib/**), то можно будет реализовать матричную факторизацию, о которой мы говорили в главе 11. Если вы хотите сделать рекомендатор на основе контента, например из главы 11, есть реализация модели LDA, но я еще не нашел подходящую книгу. Подпишитесь на мой блог (**kim-falk.org**), и скоро я выложу что-нибудь.

14.3. Что ждет рекомендательные системы в будущем?

Предсказывать будущее оказывается трудно, и запомнят только тех пророков, кто оказался прав, так что давайте надеяться, что записи в следующих разделах будут помнить. Но скорее всего, будущее будет совершенно иным.

Я точно уверен в том, что рекомендаторы будут буквально везде. Они будут реализованы на JavaScript для веб-приложений. Они будут работать на всем и помогать в любом процессе принятия решений.

Профили пользователей

В Дании было много дискуссий по поводу того, что все становится электронным и доступным только через интернет. Датское правительство перестало отправлять бумажные письма и общается только по e-mail. Однако государственная политика и ударные темпы инноваций не пришлись по вкусу всем гражданам. Это создает увеличивающийся разрыв между людьми, которые находятся в новом веке, и людьми, которые еще в прошлом.

Почему это важно в данном контексте? Люди «нового века» – это любимцы специалистов по рекомендациям, так как дают нам больше данных для работы. Сбор информации о людях, которые используют интернет, растет и растет,

и мы можем получить хорошее представление о том, что им нравится. Facebook и LinkedIn учат пользователей оформлять удобочитаемый профиль. Я думаю, что в не слишком отдаленном будущем Facebook или другая компания представят публичный машиночитаемой профиль для алгоритмов машинного обучения.

Публичный профиль позволит вашим системам получить доступ к информации и использовать эти профили, чтобы узнать побольше о клиентах. Использование списков фильмов или книг, которые нравятся человеку, будет подспорьем для индустрии развлечений. Но общественные профили также могут содержать информацию о сегменте, если у человека, например, есть машина или дети, как показано на рис. 14.4. Это тоже поможет другим типам рекомендаторов.

Подобный машиночитаемый профиль позволит людям думать о том, какой информацией стоит делиться, и стоит ли делиться вообще. Сейчас такие сайты, как Facebook, позволяют людям использовать их аккаунт для входа на разные сайты. То есть, по сути, вы даете другой компании доступ к вашему профилю[1]. В определенном смысле ваш профиль превращается в легкодоступные данные.

Рис. 14.4. Профиль пользователя может содержать информацию о том, какие фильмы и книги он любит, есть ли у него дети, авто, велосипед

[1] Многие ссылаются на Общие правила защиты данных (GDPR), и не без оснований. Но я боюсь, что новые законы не остановят сбор данных пользователей. Пользовательские соглашения станут еще длиннее, а их и без того никто не читает.

Машиночитаемой профиль также может позволить улучшить вопросы приватности, так как вы сможете решить, какую информацию раскрывать о себе, например сообщить системе, что вы недавно читали последний роман Питера Гамильтона и вам интересны новые тенденции в области органического пищевого сырья.

Для людей, которые не хотят давать доступ к информации о себе, есть компании, специализирующиеся на создании профилей пользователей. Это означает, что независимо от того, что вы делаете, сервис, скорее всего, будет что-то знать о вас с первого дня.

Наверное, будет еще труднее генерировать рекомендации для людей, которые не сидят в интернете, но которые используют офлайн-мобильные приложения. Но современные дети уже не знают времен без смартфонов и интернета, и этот сегмент, скорее всего, угаснет. В ближайшем будущем все еще останутся проблемы с «холодными» клиентами.

Контекст

Я считаю, что рекомендатели скоро будут использоваться во многих задачах. Не только потому, что почти все устройства сейчас мобильные, имеют GPS и другие инструменты, позволяющие понять текущий контекст (рис. 14.5), но и потому, что в ближайшее время все устройства будут содержать так много информации, что людям будут необходимы рекомендации для дальнейших действий. И я говорю не только о покупке вещей. Рекомендаторы также будут использоваться для принятия решений во многих других ситуациях.

Например, рекомендаторы типа «Что делать дальше» становятся все популярнее в маркетинге и банковской деятельности. Они предназначены не для конечного пользователя, а для банкиров, которые предлагают что-то клиентам. То же самое подойдет и для адвокатов, и еще много для кого.

Текущие исследования и большая часть знаний о рекомендаторах касается алгоритмов, которые делают все расчеты в автономном режиме. Это является предметом дальнейших исследований и позволяет тестировать алгоритм на тестовом наборе. Вы даже можете прибегнуть к помощи других, чтобы получить лучший результат. Как было неоднократно упомянуто в этой книге, такой подход не гарантирует получение хорошего рекомендатора. Я думаю, что будущее рекомендаторов – динамические алгоритмы, которые будут, впрочем, основаны на автономных вычислениях, о которых мы говорили. Идея усиленного обучения будет иметь гораздо более важную роль.

Другой пример контекста: ваш телефон подключен к смарт-часам, которые измеряют параметры тела и подсказывают, что вам нужно попить. Система тут же рекомендует вам кофейню за углом. Или часы замечают, что частота вашего пульса указывает на стресс, и система тут же запускает успокаивающую музыку. Это своего рода ваш персональный советник во всех делах.

Для более глубокого изучения контекстно-зависимых рекомендательных систем почитайте «Учебник по рекомедательным системам» Франческо

Риччи и др. (Springer, 2015). Глава из этой книги о контекстно-зависимых рекомендаторах доступна онлайн[1].

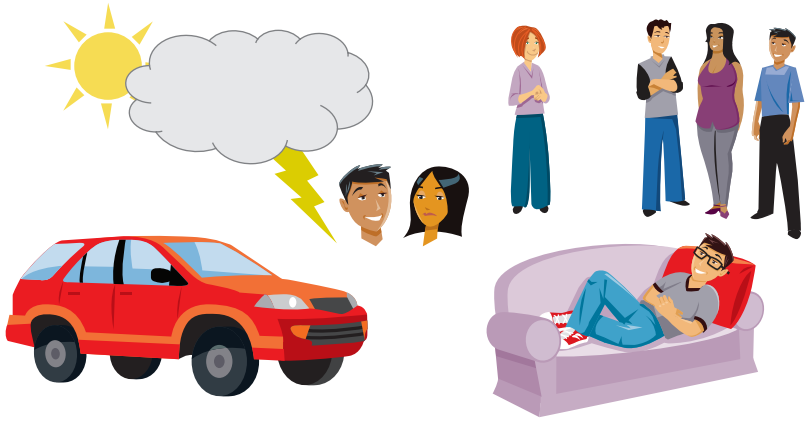


Рис. 14.5. Контекст может включать много всего: погода, настроение, наличие компании, нахождение за рулем авто или дома на диване. И все это будет означать разные вещи в разных областях

Алгоритмы

Когда я впервые начал работать с рекомендательными системами, я считал их прямой эволюцией поисковых систем. Везде было слишком много данных, и их нужно было фильтровать, прежде чем рекомендовать. С этой точки зрения рекомендаторы стали чем-то похожи на поисковики, где к запросу также прибавляются данные пользователя, настроение и многое другое.

Мохаммед Хуссейн Тагави, старший инженер-исследователь Netflix, заявил на своей презентации на RecSys2016, что идеальное состояние рекомендательной системы достигается тогда, когда вы «включаете Netflix, и играть начинает идеальный для вас контент» (**mng.bz/2152**). Но чтобы сделать это, вам нужны алгоритмы, которые смогут объединить большую модель и также включают в себя гораздо больше знаний о текущих пользователях.

В главе 12 мы говорили об ансамблях, которые могут обрабатывать больше входных данных и больше различных моделей. Опять же, глубокое обучение также рассматривается как шаг к улучшению всего. Какими будут алгоритмы – неизвестно. Возможно, вы придумаете что-то эдакое.

Конфиденциальность

В социальном интернете появляется все больше и больше данных о людях и их социальных связях. Совместная фильтрация хороша тем, что она связывает поведение разных пользователей, и в будущем люди захотят видеть рекомендации, основанные на вкусах близких и надежных друзей, а не случайной

[1] Для доступа к PDF-файлу посетите **www.researchgate.net/publication/220605653_Context-Aware_Recommender_Systems**.

группы людей с похожими вкусами. Рекомендаторы на основе доверия уже есть и набирают обороты.

Пока я перечитывал книгу, в Facebook разразился скандал с данными. Вероятно, это будет обсуждение в течение следующих нескольких лет, особенно с этими новыми европейскими законами. Я боюсь, что даже с новым законодательством люди скоро забудут весь этот шум и откажутся от приватности в пользу получения сервисов, а потом будут удивляться, что их данные продаются. Но я надеюсь, что все эти неприятности заставят людей осознать, что они должны думать о том, что выкладывать в интернет. А сервисы, надеюсь, будут немного более осторожным при обработке данных своих пользователей. Как дата-сайентист, я призываю вас использовать данные разумно и уважать частную жизнь других людей.

Архитектура

Рекомендательные системы, естественно, понадобятся, когда у пользователя будет много вариантов чего-либо. В будущем, я полагаю, множество вариантов будет везде, а не только в развлекательной сфере, как Netflix. При огромных объемах данных рекомендательные алгоритмы столкнутся с проблемами, и будет слишком сложно генерировать рекомендации, основанные на терабайтах данных. Нам нужно искать новые типы алгоритмов, которые смогут обрабатывать огромные объемы данных или, по крайней мере, оптимизировать имеющиеся алгоритмы.

Рекомендаторы будут работать везде и даже на небольших устройствах. Но с нынешней скоростью развития телефонов и других устройств вполне вероятно, что это не будет проблемой, и их вычислительных мощностей будет достаточно. Сегодняшние «большие и сложные» модели скоро будут легко умещаться в смартфон в виде небольшой локальной библиотеки.

Сюрпризы в рекомендациях

Одна из самых больших проблем рекомендательных систем заключается в том, что они редко удивляют. Нам необходимо выяснить, как лучше организовать пересечение категорий и рекомендовать вещи из каталога с максимальным охватом. Это будет одна из самых больших проблем в будущем.

Есть много предложений о том, как это может быть сделано. В книге «Метрики новизны и разброса в рекомендательных системах: выбор, открытие и релевантность» Пабло Кастельса и др. есть список различных методов для измерения новизны и разнообразия рекомендаций[1]. Но задача это непростая.

14.4. Послесловие

Придется признать, что я не читаю последние разделы в книгах по машинному обучению, так как я занят исследованиями того, что люди пишут. Большинство авторов не пишет в книге послесловий, но я кое-что напишу.

[1] Р. Кастельс и др., «Метрики новизны и разброса в рекомендательных системах: выбор, открытие и релевантность». Ссылка: **ir.ii.uam.es/rim3/publications/ddr11.pdf**.

Я встречал много людей, которые говорили, что собирались написать подобную книгу и что если бы они потрудились написать ее, то справились бы гораздо лучше.

Возможно, и так. Тем не менее большинство из этих людей так и не начали ничего писать.

Написание книги – это большое дело, в процессе которого вы много узнаете и по теме, и в искусстве написания. Кроме того, это позволит вам проверить на прочность связи с вашими близкими.

Что касается написания и рекомендательных систем, мне еще многому надо научиться. А что касается близких – тут все прекрасно, даже если мне теперь придется наверстать пару лет упущенного времени с семьей и друзьями. Если вы все же решите писать книгу на техническую тематику, то Manning – это лучший вариант.

Многие бы сказали, что написание такой книги занимает все время и не даст в ответ ничего, кроме морального удовлетворения. Это возможно. Однако в процессе написания я многое узнал, встретил много новых людей. Это было круто, и я надеюсь, что читать вам понравится и вы узнаете все, что хотели. На прощание приведу цитату:

Проблемы с технологиями настанут тогда, когда нам будет достаточно, чтобы вещь хотя бы работала.

Дуглас Адамс (2002)

Предметный указатель

Р

C

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**.

Ким Фальк

Рекомендательные системы на практике

Главный редактор *Мовчан Д. А.*
dmkpress@gmail.com
Перевод *Павлов Д. М.*
Корректор *Абросимова Л. А.*
Верстка *Луценко С. В.*
Дизайн обложки *Мовчан А. Г.*

Формат 70×100 1/16.
Гарнитура «PT Serif». Печать цифровая.
Усл. печ. л. 36,4. Тираж 200 экз.

Веб-сайт издательства: www.dmkpress.com