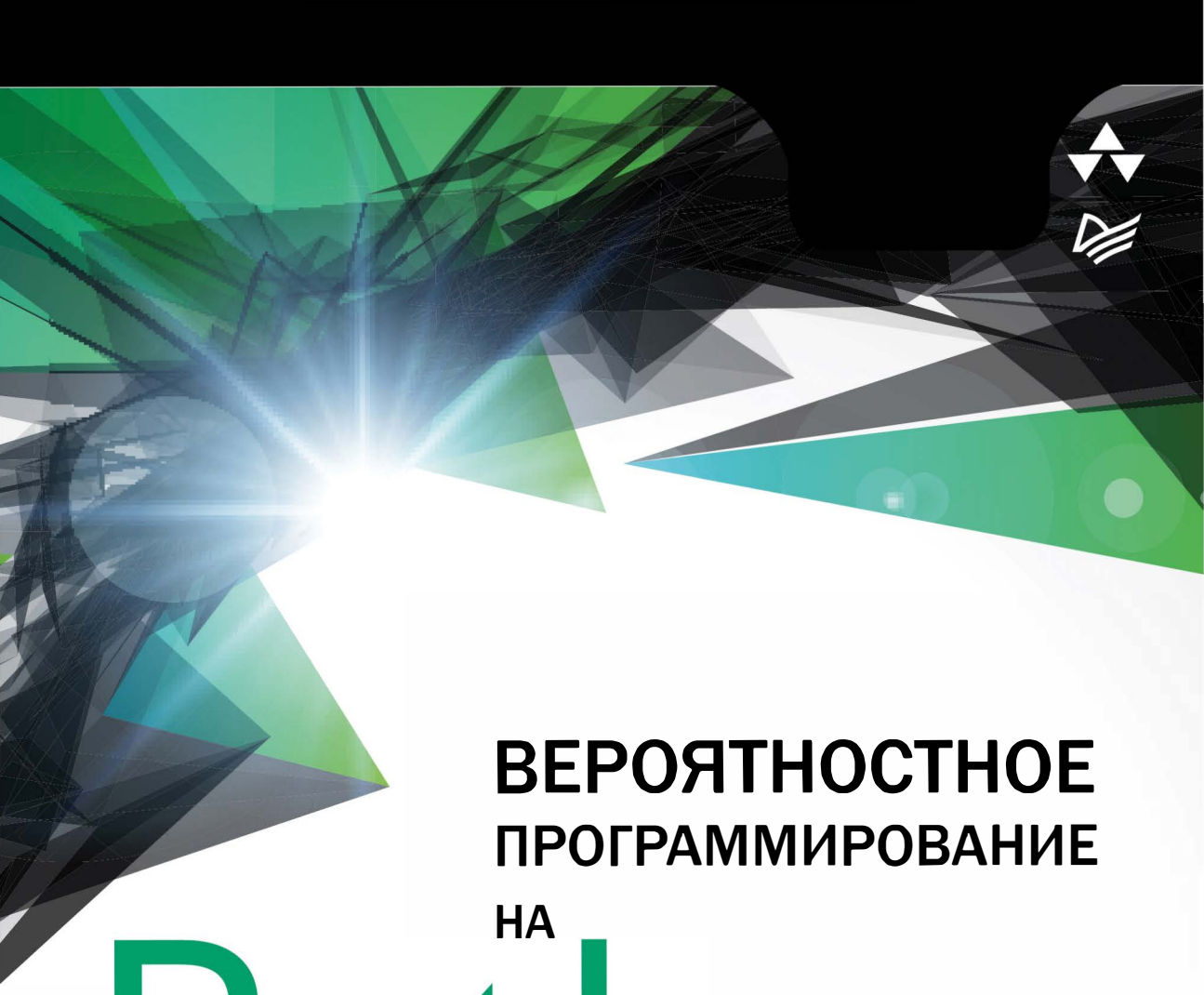

ВЕРОЯТНОСТНОЕ ПРОГРАММИРОВАНИЕ НА Python

Байесовский вывод и алгоритмы

КЭМЕРОН ДЭВИДСОН-ПАЙЛОН

CAMERON DAVIDSON-PILON

BAYESIAN METHODS FOR Hackers

Probabilistic Programming and Bayesian Inference

КЭМЕРОН ДЭВИДСОН—ПАЙЛОН

ВЕРОЯТНОСТНОЕ ПРОГРАММИРОВАНИЕ НА Python

Байесовский вывод и алгоритмы

ПИТЕР®

Санкт-Петербург • Москва • Екатеринбург • Воронеж
Нижний Новгород • Ростов-на-Дону
Самара • Минск

2019

ББК 32.973.2-018.1
УДК 004.43
Д94

Дэвидсон-Пайлон Кэмерон

Д94 Вероятностное программирование на Python: байесовский вывод и алгоритмы. — СПб.: Питер, 2019. — 256 с.: ил. — (Серия «Библиотека программиста»).

ISBN 978-5-4461-1058-2

Байесовские методы пугают формулами многих айтишников, но без анализа статистики и вероятностей сейчас не обойтись. Кэмерон Дэвидсон-Пайлон рассказывает о байесовском методе с точки зрения программиста-практика, работающего с многофункциональным языком PyMC и библиотеками NumPy, SciPy и Matplotlib. Раскрывая роль байесовских выводов при A/B-тестировании, выявлении мошенничества и в других насущных задачах, вы не только легко разберетесь в этой нетривиальной теме, но и начнете применять полученные знания для достижения своих целей.

16+ (В соответствии с Федеральным законом от 29 декабря 2010 г. № 436-ФЗ.)

ББК 32.973.2-018.1
УДК 004.43

ISBN 978-0133902839 англ.
ISBN 978-5-4461-1058-2

Краткое содержание

Оглавление

Эта книга посвящена многим важным в моей жизни отношениям: с родителями, с братьями, с ближайшими друзьями. Она также посвящена сообществу open-source, результатами труда которого мы пользуемся ежедневно, даже не подозревая об этом.

Предисловие

Байесовские методы — один из многих инструментов в арсенале современных ученых, специализирующихся на обработке данных (data scientists). Эти методы можно использовать для решения задач прогнозирования, классификации, ранжирования данных, логического вывода, обнаружения спама и многих других. Однако в большинстве существующих материалов по байесовской статистике и выводу основное внимание уделяется математическим расчетам, а не техническим темам. Поэтому я очень рад, что в нашей серии появилась книга, дающая введение в байесовские методы и ориентированная на практиков.

Благодаря знаниям Кэмерона в данной области и его стремлению привязать теорию к реальным примерам эта книга станет великолепным пособием как для ученых, специализирующихся на анализе данных, так и для обычных программистов, желающих изучить байесовские методы. Книга наполнена примерами, иллюстрациями и программным кодом на языке Python, что позволяет с легкостью начать решать реальные задачи. Если вы новичок в науке анализа данных, байесовских методах или в применении языка Python для подобных задач, эта книга послужит бесценным ресурсом для скорейшего старта.

Пол Дикс (Paul Dix),
редактор серии

Введение

Байесовский метод — естественный подход к логическому выводу, но он обычно скрыт от читателя за несколькими главами подробного математического анализа. Стандартная книга, посвященная байесовскому выводу, включает две-три главы по теории вероятности и только затем переходит к сути байесовского вывода. К сожалению, из-за сложности математической трактовки большинства байесовских моделей читателю демонстрируются лишь простейшие, искусственные примеры. В итоге после прочтения книги он задается вопросом: «И что с того?» К слову, я тоже так раньше считал.

В связи с недавними успехами байесовских методов в соревнованиях по машинному обучению я решил снова изучить предмет. Даже с моей математической подготовкой мне пришлось три дня подряд разбирать примеры, чтобы воссоздать целостную картину и понять суть этих методов. Тогда попросту не было достаточного количества литературы, связывающей теорию с практикой. Мое непонимание было вызвано разрывом между байесовской математикой и вероятностным программированием. Я помучился тогда, чтобы читателям не пришлось мучиться сейчас. Данная книга призвана сократить упомянутый разрыв.

Если байесовский вывод — это цель, то математический анализ — один из путей ее достижения. С другой стороны, вычислительные мощности сегодня настолько дешевы, что мы можем позволить себе пойти другим путем — путем вероятностного программирования. Этот путь намного более практичен, так как не требует математического вмешательства на каждом шагу, то есть мы фактически устраняем необходимость в (зачастую трудных для восприятия) математических выкладках для понимания байесовских методов. Проще говоря, вычислительный путь ведет нас к цели небольшими промежуточными шажками, а математический — гигантскими прыжками, часто уводящими далеко в сторону. Кроме того, без сильной математической подготовки попросту невозможно осуществить требуемый математический анализ.

Эта книга создавалась в качестве введения в байесовский вывод в первую очередь с алгоритмической точки зрения, с точки зрения понимания происходящего и только потом — с математической. Конечно же, с учетом того, что это вводная книга, мы остановимся только на базовых аспектах данной тематики. Математически подкованные читатели могут удовлетворить любопытство, порожденное

данной книгой, с помощью более математико-ориентированной литературы по теме. Энтузиастам с более слабой математической подготовкой, а также тем, кого интересуют не математические, а прикладные аспекты байесовских методов, данной книги будет более чем достаточно.

Выбор PyMC в качестве языка вероятностного программирования обусловлен двумя причинами. Во-первых, на момент написания этих слов не существовало центрального ресурса с примерами и объяснениями относительно PyMC. Официальная документация предполагает, что ее читатель знаком с байесовским выводом и вероятностным программированием. Надеюсь, эта книга станет стимулом попробовать PyMC для пользователей различной квалификации. Во-вторых, с учетом тенденций развития языка и в связи с популярностью математического инструментария в Python PyMC может вскоре стать компонентом его стандартной библиотеки.

Для работы PyMC понадобятся другие библиотеки, а именно NumPy и SciPy (необязательно). Дабы не ограничивать пользователя, в примерах в данной книге были задействованы библиотеки PyMC, NumPy, SciPy и matplotlib.

Порядок изложения следующий.

- Глава 1 познакомит вас с байесовским выводом и сравнит его с другими способами вывода. Мы также рассмотрим, соберем и обучим нашу первую байесовскую модель.
- Глава 2 посвящена построению моделей с использованием PyMC. Основное внимание при этом уделяется практическим примерам.
- Глава 3 познакомит вас с методом Монте-Карло на основе марковских цепей — мощным алгоритмом машинного вывода, а также с некоторыми приемами отладки байесовских моделей.
- В главе 4 я сделаю небольшое отступление и снова обращусь к проблеме объема выборки. Я объясню, почему настолько важно правильно его подбирать.
- Глава 5 познакомит вас с понятием функции потерь, когда вместо модели у вас есть функция, связывающая байесовский вывод с конкретной практической проблемой.
- В главе 6 мы вернемся к априорным распределениям в байесовском выводе, а также рассмотрим эвристики, позволяющие выбрать подходящее априорное распределение.
- Наконец, в главе 7 вы увидите, как байесовский вывод может быть использован в A/B-тестировании.

Все наборы данных, использованные в данной книге, доступны по адресу https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers.

Некоторые рисунки в книге сопровождаются QR-кодами. Воспользуйтесь любым приложением для распознавания QR-кода, чтобы увидеть иллюстрацию в цвете.

Благодарности

Я хотел бы поблагодарить людей, участвовавших в создании данной книги. В первую очередь хотелось бы сказать спасибо соавторам онлайн-версии этого издания. Многие из них внесли вклад (код, идеи, текст), который помог усовершенствовать книгу. Далее я хотел бы поблагодарить рецензентов данной книги — Роберта Мориэлло (Robert Mauriello) и Тоби Боседе (Tobi Bosede): они потратили свое время на то, чтобы продраться сквозь построенные мной сложные абстракции, и помогли сократить объем книги и обеспечить намного более приятное ее прочтение. Наконец, я хотел бы поблагодарить моих друзей и коллег, поддерживавших меня в течение всего процесса написания книги.

Об авторе

Кэмерон Дэвидсон-Пайлон (Cameron Davidson-Pilon) разбирается во многих областях прикладной математики, начиная от эволюционной динамики генов и болезней, заканчивая стохастическим моделированием финансовых рынков. Его основной вклад в сообщество open-source — книга «Вероятностное программирование на Python» и библиотека lifelines (реализующая методы анализа выживаемости, http://lifelines.readthedocs.io). Кэмерон рос в Гельфе, штат Онтарио, но образование получил в университете Уотерлу (University of Waterloo) и Независимом московском университете (НМУ). Сейчас он живет в Оттаве, штат Онтарио, и работает в интернет-компании Shopify, которая является лидером онлайн-коммерции.

От издательства

Ваши замечания, предложения, вопросы отправляйте по адресу comp@piter.com (издательство «Питер», компьютерная редакция).

Мы будем рады узнать ваше мнение!

На сайте издательства www.piter.com вы найдете подробную информацию о наших книгах.

1 Философия байесовского вывода

1.1. Введение

Вы опытный программист, но ошибки все равно проникают в ваш код. Реализовав особо трудный алгоритм, вы решили проверить его на стандартном примере. Тест пройден. Вы проверяете работоспособность кода на более сложной задаче. Снова успех. Код даже проходит следующий, *куда более сложный* тест! Вам начинает казаться, что в коде, возможно, нет ошибок...

Если это напоминает ход ваших мыслей — поздравляем, вы уже мыслите по-байесовски! Байесовский вывод, по сути, состоит в том, чтобы уточнять свое понимание по мере появления новых доказательств. Байесовец редко бывает безоговорочно уверен в результате, но его уверенность обычно высока. Точно так же, как и в приведенном выше примере, мы не можем быть уверенными в безошибочности кода на 100 %, если не проверим его на всех возможных задачах. Такое редко возможно на практике. Успешно проверив код на большом количестве задач, мы становимся более уверенными в его качестве, но уверенность эта все равно не безусловна. Байесовский вывод работает идентично: мы уточняем представление о результате, но не можем быть абсолютно *уверенными* в нем, пока не проверим все возможные альтернативы.

1.1.1. Байесовское мышление

Байесовский вывод отличается от традиционного статистического тем, что сохраняет *неопределенность*. Поначалу это кажется плохим статистическим приемом. Разве статистика не выводит *достоверность* из случайности? Чтобы преодолеть это противоречие, мы должны начать мыслить как байесовцы.

Байесовское мировоззрение интерпретирует вероятность как меру *правдоподобия события*, то есть степень нашей уверенности в том, что событие произойдет. Мы вскоре убедимся, что это и есть естественная интерпретация вероятности.

Чтобы прояснить ситуацию, рассмотрим альтернативную интерпретацию вероятности. **Фреквентисты** (от frequency — «частота»), придерживающиеся классического варианта статистики, исходят из того, что вероятность — это долгосрочная частота событий (отсюда и их название). К примеру, *вероятность авиакатастроф* в понимании фреквентистов является *долгосрочной частотой авиакатастроф*. Это логично во многих случаях, но тяжело понять, когда события не имеют долгосрочной частоты реализации. Рассмотрим следующий пример. Мы часто оцениваем вероятность результатов президентских выборов, притом что сами выборы происходят единожды. Фреквентисты обходят это ограничение, вводя параллельные миры. Вероятность, таким образом, определяется частотой проявления события во всех этих мирах.

Байесовцы, с другой стороны, следуют интуитивному подходу. Они интерпретируют вероятность как меру правдоподобия (достоверность) возникновения события. Проще говоря, вероятность — это обобщенное мнение. Индивид, приписывающий событию правдоподобие 0, абсолютно уверен в том, что событие *не* произойдет. Напротив, если он приписывает событию правдоподобие 1, то он абсолютно уверен, что событие *произойдет*. Правдоподобие между 0 и 1 позволяет задать вес других исходов. Такое определение согласуется с примером о вероятности авиакатастрофы, так как степень уверенности индивида, наблюдавшего частоту авиакатастроф и не имеющего никакой дополнительной информации, должна совпадать с частотой возникновения авиакатастроф. Аналогично согласно определению, приравнивающему вероятность к уверенности, имеет смысл говорить о вероятности исхода (уверенности в исходе) президентских выборов. Насколько вы уверены в том, что кандидат А победит?

Обратите внимание, что в предыдущем абзаце я приписал меру уверенности (вероятности) *индивиду*, а не природе. Такое определение интересно тем, что допускает конфликт мнений индивидов. Опять-таки это аналогично тому, что происходит естественным образом. Разные индивиды имеют разную степень уверенности в том, что событие произойдет, поскольку они имеют разную информацию об окружающем мире. Существование разной степени уверенности не подразумевает того, что кто-то не прав.

Рассмотрим следующие примеры, демонстрирующие соотношение между личной уверенностью и вероятностью.

1. Я подбрасываю монетку, и мы оба пытаемся угадать исход. Исходя из предположения, что монетка «честная», мы оба сходимся во мнении, что вероятность выпадения орла равна 0,5. Допустим, я подсмотрел, какой стороной упала монетка. Теперь я точно знаю, каков результат, поэтому я приписываю орлу или решке — смотря что выпадет — вероятность 1. Теперь скажите, насколько *вы* уверены в том, что выпал орел? Мое знание об исходе эксперимента не повлияло

на его результат. Следовательно, мы приписываем конкретному исходу разные вероятности.

2. В вашем коде либо есть ошибки, либо нет, но наверняка мы не знаем, хотя и имеем определенную степень уверенности относительно наличия или отсутствия ошибок.
3. У пациента проявляются симптомы x, y и z. Есть ряд заболеваний, которые могут вызывать все перечисленные симптомы, но установлено лишь одно из них. Доктор имеет определенную степень уверенности в том, какое именно. Другой доктор может иметь уверенность в наличии другого заболевания.

Представлять уверенность как вероятность естественно для человека. Взаимодействуя с миром, мы постоянно так делаем — нам видны только частные истины, но при этом мы собираем свидетельства, формирующие нашу уверенность в чем-либо. Напротив, чтобы мыслить как фреквентист, необходимо иметь определенную подготовку.

Чтобы не расходиться с традиционной нотацией, принятой в теории вероятностей, обозначим нашу уверенность относительно события A как $P(A)$. Будем называть эту величину **априорной вероятностью**.

Джону Мейнарду Кейнсу (John Maynard Keynes) часто (и, похоже, ошибочно) приписывают следующую цитату: «Когда меняются факты, я меняю мнение. А что делаете вы, сэр?» Эта цитата отражает то, как байесовец уточняет свою уверенность в чем-либо, получив новые доказательства. Даже (и в особенности) если новые доказательства противоречат текущим убеждениям, они не могут быть проигнорированы. Обозначим новое убеждение как $P(A|X)$, что будем интерпретировать как вероятность события A при наличии свидетельства X. Новое убеждение будем называть **апостериорной вероятностью** в противоположность априорной. Рассмотрим, к примеру, апостериорные вероятности (апостериорные убеждения) событий, описанных в предыдущих примерах, вычисленные после получения свидетельства X.

1. $P(A)$ — монетка с вероятностью 50 % упадет орлом вверх. $P(A|X)$ — вы смотрите на монетку, видите, что она упала орлом вверх, обозначаете эту информацию как X и приписываете выпадению орла вероятность 1, а выпадению решки — 0.
2. $P(A)$ — в большом и сложном коде, вероятно, есть ошибка. $P(A|X)$ — код прошел все тесты из набора X; в нем все еще могут быть ошибки, но их наличие теперь менее вероятно.
3. $P(A)$ — у пациента может быть сколь угодно много болезней. $P(A|X)$ — в результате анализа крови получено доказательство X, исключающее из рассмотрения некоторые болезни.

Очевидно, что в каждом из примеров, наблюдая доказательство X, мы не отклоняем предыдущее убеждение полностью, а лишь *меняем его вес*, чтобы учесть

новые доказательства (то есть мы увеличиваем или уменьшаем вес (степень достоверности) тех или иных убеждений относительно друг друга).

Вводя априорную неопределенность событий, мы тем самым сразу признаемся в том, что наши догадки могут оказаться совершенно неверными. Получив данные, доказательства либо другую информацию, мы уточняем свои убеждения, и тогда наши догадки становятся *менее ошибочными*. Фактически это другая сторона медали процесса предсказания — обычно мы пытаемся стать *более правыми*.

1.1.2. Байесовский вывод на практике

Если бы фреквентистский и байесовский выводы были функциями в программе, принимающими на входе статистические задачи, то они бы различались возвращаемыми пользователю результатами. Фреквентистская функция вывода возвращала бы число, обозначающее оценку (обычно обобщенную величину вроде выборочного среднего). Байесовская же функция вывода вернула бы множество *вероятностей*.

К примеру, в рамках задачи отладки вызов фреквентистской функции с аргументом «Мой код прошел все тесты *X*, он безошибочен?» вернул бы *ДА*. С другой стороны, спрашивая у байесовской функции: «В моем коде зачастую есть ошибки. Мой код прошел все тесты *X*, он безошибочен?» — мы получим нечто другое: вероятности ответов *ДА* и *НЕТ*. Функция может вернуть:

```
ДА с вероятностью 0,8; НЕТ с вероятностью 0,2.
```

Такой ответ сильно отличается от ответа фреквентистской функции. Обратите внимание, что байесовская функция принимает дополнительный аргумент: «В моем коде часто есть ошибки». Это априорный факт. Включая априорный параметр, мы указываем байесовской функции учесть наше мнение о ситуации. Чисто технически этот параметр байесовской функции не является обязательным, но его исключение, как мы увидим позже, имеет свои последствия.

Учет доказательств. По мере получения новых доказательств наши предыдущие убеждения как бы «вымываются» вновь поступившими результатами. Это ожидаемо. Допустим, вы верите в какую-нибудь глупость вроде «Солнце завтра взорвется» и каждый день оказываетесь не правы. Хотелось бы надеяться, что любой механизм вывода скорректирует подобные убеждения или хотя бы сориентирует их в нужном направлении. Байесовский вывод способен это сделать.

Обозначим количество имеющихся у нас экземпляров доказательств буквой *N*. По мере получения бесконечного количества доказательств ($N \to \infty$) результаты байесовского вывода (чаще всего) согласуются с результатами фреквентистского вывода. Следовательно, при больших значениях *N* статистический вывод более или менее объективен. С другой стороны, при малых значениях *N* вывод намного

более неустойчив — фреквентистские оценки имеют большую дисперсию и более широкий доверительный интервал. В таком случае более выигрышным оказывается байесовский анализ.

Вводя априорные распределения и возвращая вероятности, а не скалярные оценки, мы сохраняем неопределенность, отражающую нестабильность статистического вывода на выборках с малым *N*.

Можно предположить, что при больших значениях *N* неважно, какой подход выбрать, поскольку они оба дают схожие результаты. Такое предположение склоняет к использованию вычислительно более простых фреквентистских методов. Тому, кто так думает, стоит обратить внимание на следующее высказывание Эндрю Гельмана (Andrew Gelman) [1], прежде чем принимать поспешное решение:

> «Размер выборки никогда не бывает достаточно велик. Если N слишком мало, чтобы получить достаточно точную оценку, то нужны дополнительные данные или предположения. Если же число N "достаточно велико", можно начать делить выборку, чтобы получить дополнительные сведения (к примеру, если по результатам опроса общественного мнения получена хорошая оценка по всей стране, можно выполнить оценку среди мужчин и женщин, представителей различных возрастных групп, жителей северных и южных регионов и т. п.). N никогда не будет достаточно велико, поскольку если бы это было так, то вы бы уже решали другую проблему, для которой требуется больше данных».

1.1.3. Корректны ли фреквентистские методы?

Нет. Фреквентистские методы до сих пор полезны во многих областях знаний или даже соответствуют их современному состоянию. Существует ряд мощных и быстрых инструментов, таких как линейная регрессия методом наименьших квадратов, LASSO-регрессия и алгоритм максимизации ожидания. Байесовские методы дополняют эти приемы, решая задачи, им недоступные, либо проливая свет на лежащие в их основе системы с помощью более гибкого моделирования.

1.1.4. О проблеме больших данных

Как ни парадоксально, проблемы предиктивного анализа в области больших данных обычно решаются относительно простыми алгоритмами [2; 3]. Следовательно, можно говорить о том, что сложность прогнозирования на основе больших данных заключается не в используемом алгоритме, а скорее в вычислительной сложности хранения и обработки больших массивов данных. Стоит вспомнить вышеупомянутую цитату о размерах выборок и задаться вопросом: «Действительно ли имеющиеся у меня данные — большие?»

Во многих сложных аналитических задачах речь идет о *данных среднего размера* (medium data) или — доставляющих больше всего проблем — *по-настоящему малых данных*.

1.2. Понятийный аппарат байесовского подхода

Нас интересуют убеждения (beliefs), которые в терминах байесовского подхода могут быть интерпретированы как вероятности. У нас есть априорная уверенность в событии A, например в том, что перед выполнением тестов код содержит ошибки.

Затем мы наблюдаем очередное свидетельство. В продолжение примера с ошибочным кодом: если код проходит X тестов, то необходимо изменить нашу уверенность с учетом данного факта. Величина, характеризующая уточненную уверенность в событии, называется *апостериорной вероятностью*. Уточнение уверенности осуществляется посредством следующего уравнения, известного как теорема Байеса, названная в честь открывшего ее Томаса Байеса (Tomas Bayes):

$$P(A|X) = \frac{P(X|A)P(A)}{P(X)}$$
$$\propto P(X|A)P(A).$$

Знак $\propto$ обозначает пропорциональность. Данная формула касается не только байесовского вывода — это математический факт, используемый и вне его рамок. Байесовский вывод всего лишь использует формулу, чтобы связать априорную вероятность $P(A)$ с обновленной апостериорной вероятностью $P(A|X)$.

1.2.1. Пример: подбрасывание монетки (куда же без него)

В каждой книге по статистике должен быть пример с подбрасыванием монетки, поэтому давайте разделаемся с ним прямо сейчас. Допустим, по наивности вы не уверены в вероятности выпадения орла (спойлер: она равна 50 %). Вам кажется, что есть некое настоящее соотношение (обозначим его p), но вы не имеете априорного мнения о значении p.

Мы начинаем подбрасывать монетку и записываем результаты наблюдений: орел или решка. Это наши данные наблюдений. Стоит задаться следующим вопросом:

> «Как меняется вывод p при увеличении количества опытов?»

А точнее, как выглядят апостериорные вероятности при малом количестве данных по сравнению с большим количеством? Затем мы изображаем последовательно

уточняемые с увеличением количества данных (числа опытов) апостериорные вероятности на графике.

Апостериорные вероятности представлены кривыми, ширина которых пропорциональна неопределенности. Как показано на рис. 1.1, по мере наблюдения за данными апостериорные вероятности начинают перемещаться по числовой оси. Чем больше мы со временем накапливаем результатов наблюдений, тем более близко значения вероятностей группируются вокруг истинного значения $p = 0{,}5$, обозначенного штриховой линией.

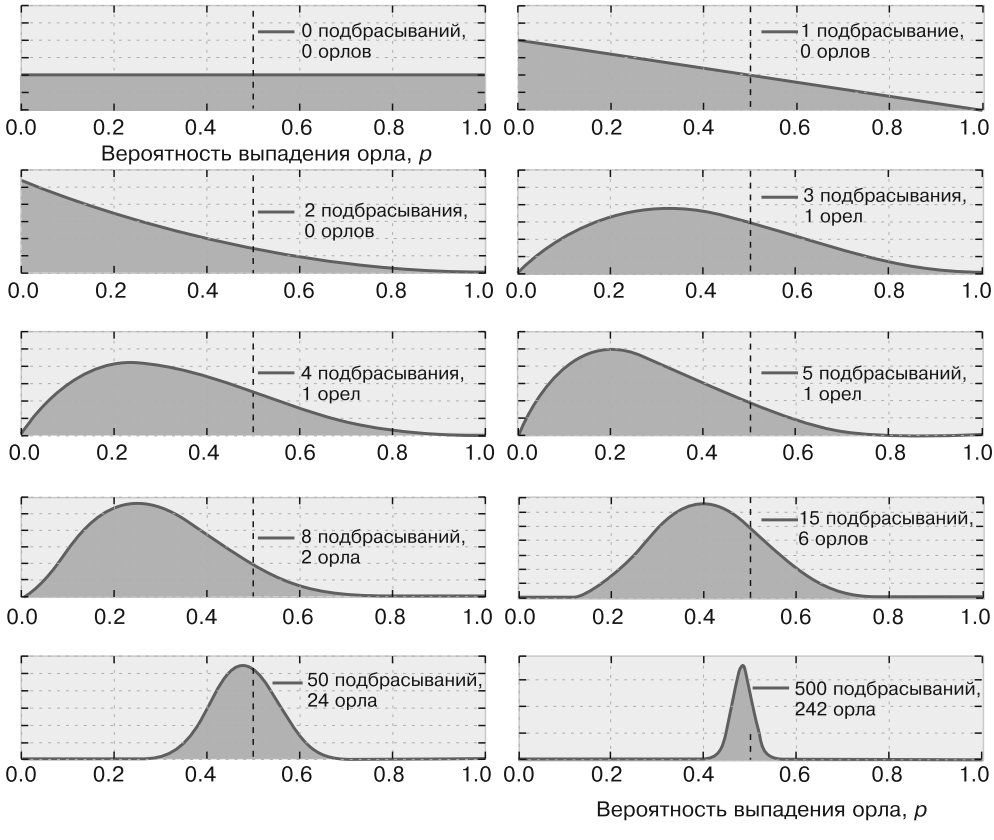

Рис. 1.1. Уточнение байесовских апостериорных вероятностей

Обратите внимание, что максимум на графике *не всегда* равен 0,5. Да и не должен быть равен — помните, что мы исходили из предположения, что у нас нет априорного мнения относительно значения p. Если бы нам попалась экстремальная выборка — скажем, один орел из восьми опытов, то наше распределение было бы сильно *смещено* относительно 0,5. (Не имея априорного мнения, насколько вы уве-

рены в том, что монета симметрична, при выпадении восьми решек и одного орла?) По мере накопления данных мы увидим, что все большее количество значений вероятности становится равным $p = 0{,}5$, но не все из них.

Следующий пример наглядно демонстрирует математику байесовского вывода.

1.2.2. Пример: библиотекарь или фермер?

Рассмотрим следующую историю (по мотивам книги Д. Канемана «Думай медленно... решай быстро») [4]. Стив — человек застенчивый, готовый помочь, но другие люди его мало интересуют. Он любит, чтобы вещи лежали на своих местах, и весьма щепетилен в своей работе. Как вы думаете, Стив скорее библиотекарь или фермер? Может показаться, что Стив скорее библиотекарь, и многие согласятся с таким выводом. Такой вывод, однако же, не учитывает «фоновое» распределение библиотекарей и фермеров. Мужчин-фермеров в 20 раз больше, чем мужчин-библиотекарей. *По статистике*, Стив скорее будет фермером!

Как мы можем исправить эту ошибку? Будет ли Стив скорее библиотекарем, нежели фермером? Предположим для простоты, что существует только две профессии — библиотекари и фермеры и что фермеров в 20 раз больше, нежели библиотекарей.

Пусть A — событие того, что Стив является библиотекарем. Если у нас нет информации о Стиве, то $P(A) = 1 / 21 = 0{,}047$. Это наше априорное распределение. Теперь предположим, что сосед Стива дал нам информацию о его личностных качествах. Обозначим эту информацию через X. Нас интересует $P(A|X)$. Вспомним теорему Байеса:

$$P(A|X) = \frac{P(X|A)P(A)}{P(X)}.$$

Нам известна вероятность $P(A)$, но что означает $P(X|A)$? Эта величина может быть определена как вероятность такого описания личности при условии, что Стив является библиотекарем. То есть насколько вероятно, что сосед опишет Стива таким образом, если он и правда работает библиотекарем. Вероятность этого близка к 1,0. Скажем, она равна 95 %, или 0,95.

Кроме того, у нас есть $P(X)$ — вероятность того, что некоторого (любого!) человека можно описать так же, как сосед описал Стива. Ее довольно сложно оценить в текущем виде, поэтому применим некоторые логические преобразования:

$$P(X) = P(X \text{ и } A) + P(X \text{ и } \sim A) = P(X|A)P(A) + P(X|\sim A)P(\sim A),$$

где $\sim A$ означает, что Стив не библиотекарь, а стало быть, он фермер. Итак, нам известны $P(X|A)$, $P(A)$ и $P(\sim A) = 1 - P(A) = 20 / 21$. Осталось найти $P(X|\sim A)$ — вероятность

того, что сосед опишет Стива как X с учетом того, что он фермер. Допустим, она равна 0,5, следовательно, $P(X) = 0{,}95 \times (1 / 21) + (0{,}5) \times (20 / 21) = 0{,}52$.

Подставляем все в исходную формулу и получаем:

$$P(A \mid X) = \frac{0{,}951 / 21}{0{,}52} = 0{,}087.$$

Не так уж и много, но с учетом того, что фермеров намного больше, чем библиотекарей, полученное значение выглядит логично. На рис. 1.2 приведено сравнение априорных и апостериорных вероятностей того, что Стив будет библиотекарем, и того, что Стив будет фермером.

```
%matplotlib inline
from IPython.core.pylabtools import figsize
import numpy as np
from matplotlib import pyplot as plt
figsize(12.5, 4)
plt.rcParams['savefig.dpi'] = 300
plt.rcParams['figure.dpi'] = 300

colors = ["#348ABD", "#A60628"]
prior = [1/21., 20/21.]
posterior = [0.087,1-0.087]
plt.bar([0, .7], prior, alpha=0.70, width=0.25,
        color=colors[0], label="априорное распределение",
        lw="3", edgecolor="#348ABD")

plt.bar([0+0.25, .7+0.25], posterior, alpha=0.7,
        width=0.25, color=colors[1],
        label="апостериорное распределение",
        lw="3", edgecolor="#A60628")

plt.xticks([0.20, 0.95], ["Библиотекарь", "Фермер"])
plt.title("Априорные и апостериорные вероятности\
          рода занятий Стива")
plt.ylabel("Probability")
plt.legend(loc="upper left");
```

Обратите внимание, что после наблюдения нами X вероятность того, что Стив библиотекарь, выросла. Ненамного, конечно: вероятность того, что Стив — фермер, все еще чрезвычайно велика.

Я привел простейший пример, иллюстрирующий байесовское правило и байесовский вывод. К сожалению, математические выкладки, требуемые для более сложного байесовского вывода, еще сложнее этих. Позже мы увидим, что в подобном математическом анализе нет необходимости. Сначала нужно расширить инструментарий моделирования.

В следующем разделе рассматриваются *вероятностные распределения*. Если вы уже знакомы с ними, можете не стесняясь пропустить этот раздел (или про-

листать). Тем, кто ими владеет в меньшей степени, просто необходимо прочесть следующий раздел.

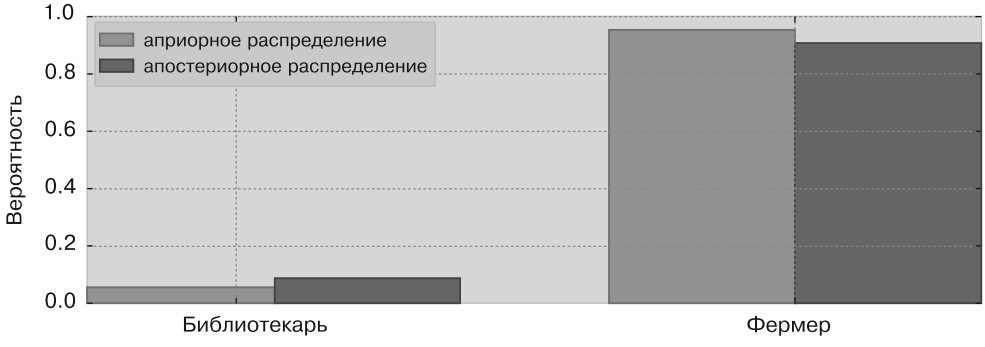

Рис. 1.2. Априорные и апостериорные вероятности рода занятий Стива

1.3. Распределения вероятностей

Давайте сначала определимся с названиями нескольких букв греческого алфавита, чтобы говорить на одном языке:

- α — альфа (alpha);
- β — бета (beta);
- λ — лямбда (lambda);
- μ — мю (mu);
- σ — сигма (sigma);
- τ — тау (tau).

Замечательно. Теперь займемся распределениями вероятностей. Напомню, что это такое. Пусть Z — некая случайная переменная (величина). С Z связана *функция распределения вероятности* (probability distribution function), задающая вероятности различных значений, принимаемых Z.

Случайные переменные можно разделить на три категории.

- **Дискретные** (discrete). Дискретные случайные переменные могут принимать значения только из заданного списка. Валюты, рейтинги фильмов и число голосов — это все дискретные случайные переменные. Смысл дискретных случайных переменных становится более понятным в сравнении с непрерывными.
- **Непрерывные** (continuous). Непрерывные случайные переменные могут принимать произвольные точные значения. Например, моделирование температуры, скорости и времени производится с использованием непрерывных случайных

переменных, поскольку значения в этом случае можно указывать все более и более точно.

- **Смешанные** (mixed). Смешанные случайные переменные содержат как дискретную, так и непрерывную часть, то есть представляют собой сочетание первых двух категорий.

1.3.1. Дискретный случай

Если случайная переменная Z — дискретная, то ее распределение называется *функцией распределения масс* (probability mass function), описывает вероятность принятия переменной Z значения k и обозначается $P(Z = k)$. Отмечу, что функция распределения масс полностью описывает случайную переменную Z, то есть по известной функции распределения масс можно сказать, как будет вести себя случайная переменная Z. Существуют часто используемые функции распределения масс, и я буду знакомить вас с ними по мере необходимости. Начнем мы с одной чрезвычайно полезной функции распределения масс. Говорят, что случайная переменная Z распределена *по закону Пуассона*, если:

$$P(Z = k) = \frac{\lambda^k e^{-\lambda}}{k!}, \quad \text{k} = 0, 1, 2...$$

где λ называется *параметром* распределения и отвечает за форму распределения. При распределении Пуассона λ может принимать любое положительное значение. При увеличении λ повышается вероятность принятия случайной переменной больших значений, и, наоборот, при уменьшении λ повышается вероятность принятия случайной переменной меньших значений. Параметр λ можно назвать *интенсивностью* распределения Пуассона.

В отличие от параметра λ, который может быть любым положительным числом, k в предыдущей формуле должно быть неотрицательным целым числом, то есть должно принимать значения 0, 1, 2 и т. д. Это очень важно, поскольку, например, при моделировании демографических показателей бессмысленно говорить о населении в количестве 4,25 или 5,612 человека.

Распределение случайной переменной Z по закону Пуассона обозначается следующим образом:

$$Z \sim \text{Poi}(\lambda).$$

Одно полезное свойство распределения Пуассона заключается в том, что его параметр равен его математическому ожиданию. То есть:

$$E[Z|\lambda] = \lambda.$$

Запомните это свойство — мы будем часто его использовать. На рис. 1.3 построен график распределения масс для различных значений λ. Прежде всего стоит

отметить, что при увеличении параметра λ повышается вероятность принятия случайной переменной бо́льших значений. Во-вторых, отмечу, что на 15 заканчивается ось *X*, но не распределение. Существует положительная вероятность выпадения каждого неотрицательного целого числа[1].

```
figsize(12.5, 4)

import scipy.stats as stats
a = np.arange(16)
poi = stats.poisson
lambda_ = [1.5, 4.25]
colors = ["#348ABD", "#A60628"]

plt.bar(a, poi.pmf(a, lambda_[0]), color=colors[0],
        label="$\lambda = %.1f$" % lambda_[0], alpha=0.60,
        edgecolor=colors[0], lw="3")

plt.bar(a, poi.pmf(a, lambda_[1]), color=colors[1],
        label="$\lambda = %.1f$" % lambda_[1], alpha=0.60,
        edgecolor=colors[1], lw="3")

plt.xticks(a + 0.4, a)
plt.legend()
plt.ylabel(u"Вероятность $k$")
plt.xlabel(u"$k$")
plt.title(u"Функция распределения масс пуассоновской случайной переменной\
          при различных значениях $\lambda$ values");
```

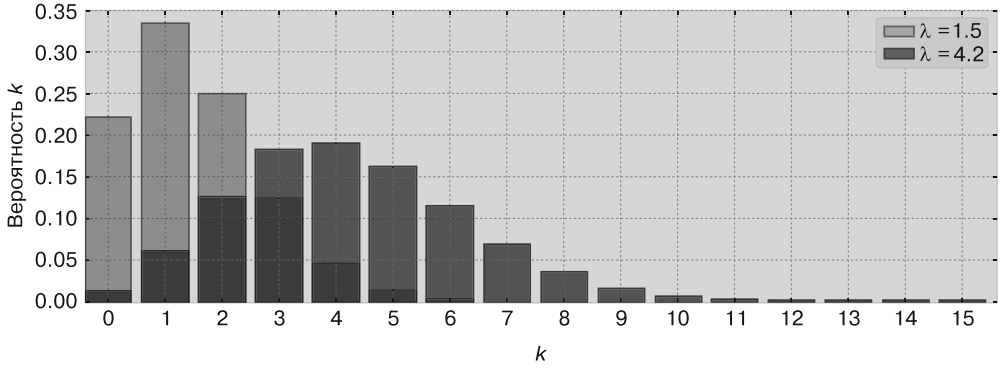

Рис. 1.3. Функция распределения масс пуассоновской случайной переменной при различных значениях λ

[1] Существует несколько способов настройки библиотеки matplotlib для правильной работы с кириллическими символами в отображаемых надписях. Простейший способ — использовать строки в кодировке Unicode. Например, для одной из строк следующего примера кода: plt.ylabel(u"Вероятность k"). — *Здесь и далее примеч. пер.*

1.3.2. Непрерывный случай

Вместо функции распределения масс, как у дискретных случайных переменных, для непрерывных случайных переменных используется *функция плотности распределения вероятности* (probability density function). Хотя введение дополнительного термина может показаться излишним, функция распределения плотности и функция распределения масс — совершенно разные вещи. Примером непрерывной случайной переменной может послужить случайная переменная с *экспоненциальной плотностью распределения вероятности*. Функция плотности для экспоненциальной случайной переменной имеет вид:

$$f_Z(z|\lambda) = \lambda e^{-\lambda z}, z \geq 0.$$

Аналогично случайной переменной с пуассоновским законом распределения экспоненциальная случайная переменная может принимать только неотрицательные значения. Но, в отличие от пуассоновской случайной переменной, экспоненциальная может принимать *любые* неотрицательные значения, включая дробные, например 4,25 или 5,612 401. Из-за этой особенности они плохо подходят для дискретных целочисленных данных, но отлично подходят для временны*х* данных, данных о температуре (измеренной в градусах Кельвина, разумеется) и любых других переменных, принимающих точные *и положительные* значения. На рис. 1.4 приведены две функции плотности вероятности с различными значениями λ.

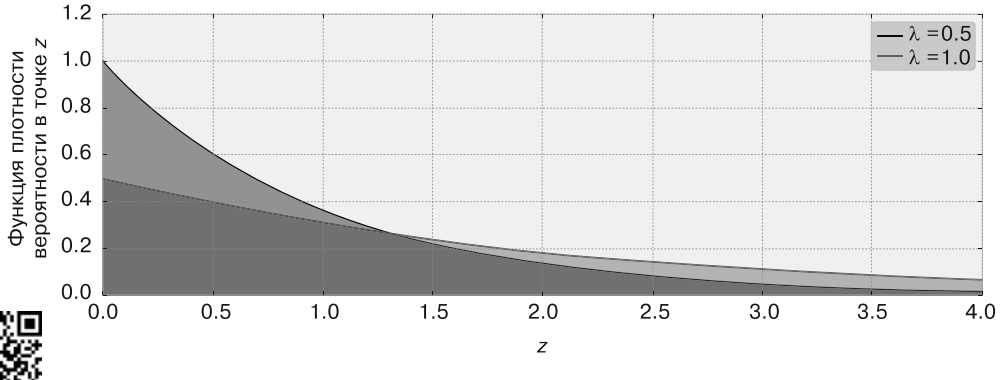

Рис. 1.4. Функция плотности вероятности экспоненциальной случайной переменной при различных значениях λ

О случайной переменной Z с экспоненциальным распределением и параметром λ говорят, что она *экспоненциальная*, и это записывается следующим образом:

$$Z \sim \text{Exp}(\lambda).$$

При заданном параметре λ математическое ожидание экспоненциальной случайной переменной обратно λ. То есть:

$$E[Z|\lambda] = \frac{1}{\lambda}.$$

```
a = np.linspace(0, 4, 100)
expo = stats.expon
lambda_ = [0.5, 1]

for l, c in zip(lambda_, colors):
    plt.plot(a, expo.pdf(a, scale=1./l), lw=3,
             color=c, label="$\lambda = %.1f$" % l)
    plt.fill_between(a, expo.pdf(a, scale=1./l), color=c, alpha=.33)

plt.legend()
plt.ylabel(u"Функция плотности вероятности в точке $z$")
plt.xlabel("$z$")
plt.ylim(0,1.2)
plt.title(u"Функция плотности вероятности экспоненциальной \
            случайной переменной при различных значениях $\lambda$ \");
```

Важно понимать, что значение функции плотности вероятности в заданной точке *не* равно вероятности этой точки. Мы поговорим об этом позднее, но, если вы хотите углубиться в этот вопрос уже сейчас, почитайте следующее обсуждение: http://stats.stackexchange.com/questions/4220/a-probability-distribution-value-exceeding-1-is-ok.

1.3.3. Но что такое λ?

Этот вопрос очень важен для статистики. На практике параметр λ от нас скрыт. Для наблюдения доступна только Z, так что для определения λ приходится проводить обратные вычисления. Сложность данной задачи состоит в отсутствии взаимно однозначного соответствия между Z и λ. Для решения задачи оценки λ было создано множество методов, но поскольку λ фактически никогда не наблюдается, то сказать с уверенностью, какой из них лучший, невозможно!

При байесовском выводе речь идет о *степени уверенности* в значении λ. Вместо того чтобы точно определить λ, мы можем говорить лишь о возможном значении λ, задавая для него вероятностное распределение.

На первый взгляд это кажется странным. В конце концов, значение параметра λ фиксированно; он вовсе не (обязательно) случайная переменная! Как можно задавать вероятности для значений неслучайной переменной? О, мы снова попались в ловушку старого частотного подхода. Напомню, что при байесовском подходе задавать значения вероятности *можно*, если интерпретировать ее как степень уверенности. А быть уверенным в той или иной степени в значении параметра λ вполне допустимо.

1.4. Использование компьютеров для автоматического байесовского вывода

Попробуем смоделировать более интересный пример, связанный с частотой отправки/получения пользователем текстовых сообщений.

1.4.1. Пример: вывод поведения на основе данных по обмену текстовыми сообщениями

Пусть дано несколько ежедневных текстовых сообщений пользователя системы. График этих данных по дням приведен на рис. 1.5. Нас интересует, менялся ли характер обмена сообщениями со временем, постепенно или резко. Как смоделировать эту ситуацию? (На самом деле это мои собственные текстовые сообщения. Если хотите, можете сделать выводы о моей популярности.)

```
figsize(12.5, 3.5)
count_data = np.loadtxt("data/txtdata.csv")
n_count_data = len(count_data)
plt.bar(np.arange(n_count_data), count_data, color="#348ABD")
plt.xlabel(u"Время (дни)")
plt.ylabel(u"Получено текстовых сообщений")
plt.title(u"Менялся ли характер обмена сообщениями пользователя \n
          со временем?")
plt.xlim(0, n_count_data);
```

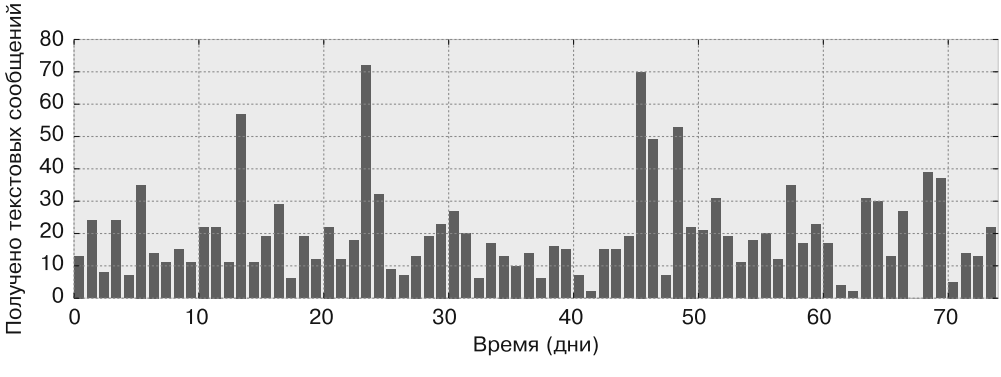

Рис. 1.5. Менялся ли характер обмена сообщениями пользователя со временем?

Прежде чем приступить к моделированию, скажите, какие выводы вы можете сделать просто из рис. 1.5? Менялось ли со временем поведение пользователя, на ваш взгляд?

С чего начать моделирование? Ну, как мы уже видели, для моделирования подобных *счетных* данных отлично подходит пуассоновская случайная переменная. Обозначим C_i текстовые сообщения за день i:

$$C_i \sim \text{Poi}(\lambda).$$

Нам неизвестно точное значение параметра λ. Из рис. 1.5 складывается впечатление, что частота сообщений повышается в течение периода наблюдений, что эквивалентно предположению об увеличении параметра λ в какой-то момент этого периода. Напомню, что более высокое значение λ говорит о повышении вероятности бо́льших значений. То есть вероятность отправки большого количества текстовых сообщений в заданный день выше.

Как описать эти наблюдения математически? Допустим, в какой-то день в течение периода наблюдений (назовем его τ) параметр λ резко возрастает. Таким образом, у нас есть фактически два параметра λ: один — для периода до τ, другой — для оставшейся части периода наблюдений. В литературе подобное резкое изменение значения называется *точкой ветвления* (switchpoint):

$$\lambda = \begin{cases} \lambda_1, \text{ если } t < \tau \\ \lambda_2, \text{ если } t \geq \tau \end{cases}.$$

Если же на деле никакого резкого изменения не происходит и $\lambda_1 = \lambda_2$, то апостериорные распределения двух λ должны выглядеть практически одинаково.

Нам нужно вывести неизвестные параметры λ. Чтобы воспользоваться байесовским выводом, следует задать априорные вероятности для различных возможных значений λ. Какие априорные распределения вероятностей хорошо подойдут для λ_1 и λ_2? Вспоминаем, что λ может принимать любое положительное значение. Как вы видели ранее, у *экспоненциального* распределения для положительных чисел непрерывная функция плотности, так что оно вполне подойдет для моделирования λ_i. Но не забывайте, что у экспоненциального распределения есть свой параметр, который тоже нужно включить в модель. Назовем этот параметр α.

$$\lambda_1 \sim \text{Exp}(\alpha),$$
$$\lambda_2 \sim \text{Exp}(\alpha).$$

Параметр α называется *гиперпараметром* (hyperparameter) или *переменной-предком* (parent variable). Фактически это параметр, влияющий на другие параметры.

Начальное значение α не слишком сильно влияет на модель, так что мы относительно свободны в выборе. Я предложил бы задать его равным величине, обратной выборочному среднему счетных данных. Почему? В силу моделирования λ на основе экспоненциального распределения с помощью вышеупомянутого равенства для математического ожидания можно получить следующее:

$$\frac{1}{N}\sum_{i=0}^{N} C_i \approx E[\lambda \mid \alpha] = \frac{1}{\alpha}.$$

При использовании этого значения мы выражаем не слишком большую уверенность в априорном распределении и тем самым снижаем влияние гиперпараметра. Другой вариант, который я рекомендую вам попробовать, состоит в использовании двух априорных распределений, по одному для каждого λ_i. Два экспоненциальных распределения с различными значениями α отразили бы нашу априорную уверенность в изменении частоты отправки сообщений в какой-то момент в течение периода наблюдений.

А как же насчет τ? Вследствие зашумленности данных выбрать априорное значение τ непросто. Вместо этого можно поставить в соответствие возможным дням *равномерное априорное распределение степени уверенности* (uniform prior belief). Это эквивалентно:

$$\tau \sim \text{DiscreteUniform}(1, 70)$$
$$\Rightarrow P(\tau = k) = 1 / 70.$$

Как же после всего вышесказанного будет выглядеть наше общее априорное распределение для неизвестных величин? Откровенно говоря, *оно не имеет значения*. Достаточно знать, что это безобразная мешанина символов, которая может понравиться только математику. А по мере усложнения моделей оно станет еще запутаннее. В любом случае нас фактически интересует только апостериорное распределение.

Далее мы займемся PyMC[1], библиотекой языка Python для байесовского анализа, для которой не страшен сотворенный нами математический монстр.

1.4.2. Наш первый инструмент: PyMC

PyMC — библиотека языка Python для байесовского анализа [5] — быстрая, с хорошей поддержкой. Единственный ее недостаток — пробелы в документации по отдельным вопросам, особенно тем, в которых остро проявляются различия

[1] Сейчас уже существует новая версия этой библиотеки, PyMC 3. На сайте автора можно найти большую часть кода данной книги в портированном под PyMC 3 (а также под TensorFlow Probability) виде.

между новичками и бывалыми экспертами. Одна из задач моей книги как раз и состоит в решении этой проблемы, а также в демонстрации замечательных возможностей PyMC.

Выполним моделирование вышеописанной задачи с помощью PyMC. Подобный стиль программирования называется *вероятностным программированием* (probabilistic programming). Но это исключительно неудачное название, которое наводит на мысли о сгенерированном случайным образом коде и, вероятно, смущает и отпугивает пользователей. Код отнюдь не случаен, вероятностный он в смысле создания вероятностных моделей с помощью переменных языка программирования в качестве компонентов модели. Компоненты модели являются полноправными примитивами в фреймворке PyMC.

У Кронина [6] приводится весьма вдохновляющее описание вероятностного программирования:

> «В отличие от обычной программы, выполняемой только в одном направлении, вероятностная программа выполняется как в прямом, так и в обратном направлении. При прямом проходе делаются выводы из допущений, принятых об окружающем мире (то есть представляемое ею пространство моделей), а при обратном проходе ("от данных") ограничиваются возможные толкования. На практике во многих системах вероятностного программирования эти прямые и обратные операции продуманно переплетаются для эффективного поиска наилучших толкований».

Раз уж термин *«вероятностное программирование»* вызывает столько недоразумений, я воздержусь от его использования. Вместо него буду говорить просто *«программирование»*, чем, по существу, оно и является.

Читать код PyMC очень просто. Немного в новинку вам может показаться только синтаксис, так что я буду прерывать код для объяснения отдельных фрагментов. Просто помните, что мы представляем компоненты модели (τ, λ_1, λ_2) в виде переменных:

```
import pymc as pm

alpha = 1.0/count_data.mean() # Напомню, что count_data – переменная,
                              # хранящая количество текстовых сообщений
lambda_1 = pm.Exponential("lambda_1", alpha)
lambda_2 = pm.Exponential("lambda_2", alpha)

tau = pm.DiscreteUniform("tau", lower=0, upper=n_count_data)
```

В этом коде мы создали переменные PyMC, соответствующие λ_1 и λ_2. Мы присвоили их *стохастическим переменным* (stochastic variables) PyMC, получившим такое название потому, что прикладная часть рассматривает их как генераторы случайных чисел. Для демонстрации этого можно вызвать их встроенные методы

`random()`. Во время этапа обучения мы найдем более подходящие значения для переменной `tau`.

```
print "Random output:", tau.random(), tau.random(), tau.random()
```

```
[Output]:

Random output: 53 21 42
```

```
@pm.deterministic
def lambda_(tau=tau, lambda_1=lambda_1, lambda_2=lambda_2):
     out = np.zeros(n_count_data)   # количество точек данных
     out[:tau] = lambda_1   # lambda до дня tau равна lambda_1
     out[tau:] = lambda_2   # lambda после дня tau (и включая его) равна lambda_2


     return out
```

В этом коде[1] создается новая функция `lambda_`, которую на самом деле можно считать случайной переменной — вышеупомянутой случайной переменной λ. Замечу, что в силу случайности переменных `lambda_1`, `lambda_2` и `tau` переменная `lambda_` также будет случайной. Пока мы *не* фиксируем значений никаких переменных.

`@pm.deterministic` — декоратор, указывающий PyMC, что эта функция — детерминистичная. То есть при детерминистичных аргументах (а в данном случае это не так) возвращаемый функцией результат также будет детерминистичным.

```
observation = pm.Poisson("obs", lambda_, value=count_data, observed=True)

model = pm.Model([observation, lambda_1, lambda_2, tau])
```

В переменной `observation` наши данные, `count_data`, сочетаются посредством ключевого слова `value` с предлагаемой схемой генерации данных, которая задается переменной `lambda_`. Параметр `observed=True` указывает PyMC, что значение переменной `observation` не должно меняться в ходе нашего анализа. И наконец, мы собираем все интересующие нас переменные и создаем на их основе экземпляр `Model`. Это облегчит извлечение результатов.

Объяснения к следующему коду вы найдете в главе 3, но я привожу его здесь, чтобы было понятно, откуда взялись получившиеся результаты. Его можно считать этапом *обучения*. Используемый математический аппарат называется *методом Монте-Карло по схеме марковской цепи* (Markov Chain Monte-Carlo, MCMC), о нем мы также поговорим в главе 3. Данный подход возвращает тысячи случайных переменных из апостериорных распределений λ_1, λ_2 и τ. Чтобы посмотреть, как выглядят эти апостериорные распределения, можно построить гистограмму. Далее

[1] Здесь и далее синтаксис функции print был изменен с Python 2 на такой, который будет работать как в Python 2, так и в Python 3. Было: print "Random output:", tau.random(), tau.random(), tau.random(). Стало: print("Random output:", tau.random(), tau.random(), tau.random()).

мы собираем выборки (в литературе по MCMC называемые *следами* (traces)) в гистограммы. Результаты показаны на рис. 1.6.

```
# Таинственный код, который я поясню в главе 3.
# Достаточно сказать, что мы получим 30 000 (40 000 минус 10 000) выборок.
mcmc = pm.MCMC(model)
mcmc.sample(40000, 10000)
```

```
[Output]:

[-----------------100%-----------------] 40000 of 40000 complete in 9.6 sec
```

```
lambda_1_samples = mcmc.trace('lambda_1')[:]
lambda_2_samples = mcmc.trace('lambda_2')[:]
tau_samples = mcmc.trace('tau')[:]

figsize(14.5, 10)
# Гистограмма выборок

ax = plt.subplot(311)
ax.set_autoscaley_on(False)

plt.hist(lambda_1_samples, histtype='stepfilled', bins=30, alpha=0.85,
         label=u"апостериорное распределение $\lambda_1$", color="#A60628",
         normed=True)
plt.legend(loc="upper left")
plt.title(r"""Апостериорные распределения параметров \
          $\lambda_1,\;\lambda_2,\;\tau$""")
plt.xlim([15, 30])
plt.xlabel(u"Значение $\lambda_1$")
plt.ylabel(u"Плотность")

ax = plt.subplot(312)
ax.set_autoscaley_on(False)
plt.hist(lambda_2_samples, histtype='stepfilled', bins=30, alpha=0.85,
         label=u"апостериорное распределение $\lambda_2$", color="#7A68A6",
         normed=True)
plt.legend(loc="upper left")
plt.xlim([15, 30])
plt.xlabel(u"Значение $\lambda_2$")
plt.ylabel(u"Плотность")

plt.subplot(313)
w = 1.0 / tau_samples.shape[0] * np.ones_like(tau_samples)
plt.hist(tau_samples, bins=n_count_data, alpha=1,
         label=r"апостериорное распределение $\tau$", color="#467821",
         weights=w, rwidth=2.)
plt.xticks(np.arange(n_count_data))
plt.legend(loc="upper left")
plt.ylim([0, .75])
plt.xlim([35, len(count_data)-20])
plt.xlabel(r"$\tau$ (в днях)")
plt.ylabel(u"Вероятность");
```

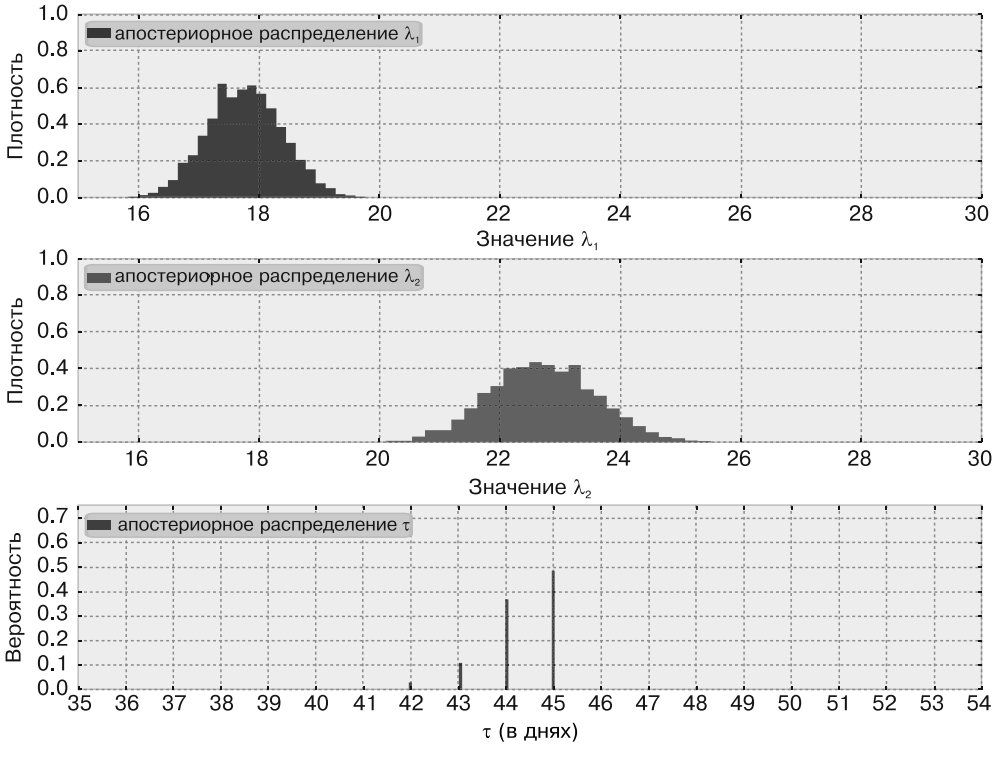

Рис. 1.6. Апостериорные распределения параметров λ_1, λ_2 и τ

1.4.3. Толкование результатов

Напомню, что в результате применения байесовской методики получается *распределение*. Следовательно, у нас теперь есть распределения, описывающие неизвестные λ и τ. Что это нам дает? Во-первых, в наших оценках сразу же видна неопределенность: чем шире распределение, тем меньше должна быть наша апостериорная уверенность. Далее мы видим подходящие значения для параметров: λ_1 — около 18, а λ_2 — около 23. Апостериорные распределения совершенно явно различаются, указывая на вполне вероятное изменение характера обмена текстовыми сообщениями.

Какие еще выводы можно сделать? Обоснованными ли выглядят эти результаты в свете исходных данных?

Отмечу также, что апостериорные распределения для параметров λ не похожи на экспоненциальные, хотя априорные для соответствующих величин были экспоненциальными. На самом деле форма апостериорных распределений не похожа на что-либо из начальной модели. Но это нормально! Это как раз одно из достоинств

точки зрения, основанной на вычислениях. При чисто математическом анализе мы пришли бы к невыразимому аналитически (и очень запутанному) распределению. А при вычислительном подходе возможность аналитического выражения для нас неважна.

В результате нашего анализа мы получили также распределение для τ. Его апостериорное распределение выглядит несколько иначе, чем два остальных, поскольку τ — дискретная случайная переменная и вероятности по интервалам для нее не задаются. Как видим, в районе 45-го дня вероятность изменения характера обмена сообщениями составляет 50 %. Если бы такого изменения не произошло или оно оказалось более плавным, то апостериорное распределение τ получилось бы размазанным в силу того, что многие дни потенциально подходили бы в качестве возможного τ. В то же время из фактических результатов видно, что только три или четыре дня хоть как-то подходят на роль точки ветвления.

1.4.4. Какую пользу могут принести выборки из апостериорного распределения?

Мы будем пытаться ответить на вынесенный в заголовок вопрос на протяжении всей оставшейся части книги, и будет преуменьшением сказать, что это даст потрясающие результаты. А пока что в завершение этой главы приведу еще один пример.

Мы воспользуемся выборками из апостериорного распределения для ответа на следующий вопрос: каково ожидаемое количество текстовых сообщений в день t, $0 \le t \le 70$? Напомню, что математическое ожидание пуассоновской величины равно ее параметру λ. Следовательно, вышеприведенный вопрос можно переформулировать следующим образом: каково ожидаемое значение λ в момент t?

В следующем коде пусть i будет индексом для выборок из апостериорных распределений. Для заданного дня t мы усредняем значение по всем возможным λ_i, где $\lambda_i = \lambda_{1,i}$ при $t < \tau_i$ (то есть до изменения поведения пользователя) и $\lambda_i = \lambda_{2,i}$ в противном случае.

```
figsize(12.5, 5)
# tau_samples, lambda_1_samples, lambda_2_samples содержат
# N выборок из соответствующего апостериорного распределения.
N = tau_samples.shape[0]
expected_texts_per_day = np.zeros(n_count_data) # Число точек данных
for day in range(0, n_count_data):
    # ix — булев индекс всех выборок tau, соответствующих точке
    # ветвления, имевшей место ранее для значения "дня"
    ix = day < tau_samples
    # Каждая апостериорная выборка соответствует какому-то значению tau.
    # Для каждого дня это значение tau указывает, находимся ли мы "до"
    # (в режиме lambda_1) или "после" (в режиме lambda_2) точки ветвления.
    # Путем взятия апостериорных выборок соответственно lambda_1/2
    # можно посредством усреднения по всем выборкам
```

```
    # получить математическое ожидание lambda для конкретного дня.
    # Как я уже объяснял, случайная переменная для "числа сообщений"
    # распределена по закону Пуассона, а значит, lambda (параметр закона
    # Пуассона) равен математическому ожиданию "числа сообщений".
    Expected_texts_per_day[day] = (lambda_1_samples[ix].sum()
                                   + lambda_2_samples[~ix].sum()) / N

plt.plot(range(n_count_data), expected_texts_per_day, lw=4,
         color="#E24A33", label=u"Ожидаемое число полученных сообщений")
plt.xlim(0, n_count_data)
plt.xlabel(u"День")
plt.ylabel(u"Количество текстовых сообщений ")
plt.title(u"Число полученных текстовых сообщений по сравнению с ожидаемым")
plt.ylim(0, 60)
plt.bar(np.arange(len(count_data)), count_data, color="#348ABD",
        alpha=0.65, label=u"зарегистрированное число текстовых сообщений в день")
plt.legend(loc="upper left")
print expected_texts_per_day
```

```
[Output]:

[ 17.7707 17.7707 17.7707 17.7707 17.7707 17.7707 17.7707 17.7707
  17.7707 17.7707 17.7707 17.7707 17.7707 17.7707 17.7707 17.7707
  17.7707 17.7707 17.7707 17.7707 17.7707 17.7707 17.7707 17.7707
  17.7707 17.7707 17.7707 17.7707 17.7707 17.7707 17.7707 17.7707
  17.7707 17.7707 17.7707 17.7708 17.7708 17.7712 17.7717 17.7722
  17.7726 17.7767 17.9207 18.4265 20.1932 22.7116 22.7117 22.7117
  22.7117 22.7117 22.7117 22.7117 22.7117 22.7117 22.7117 22.7117
  22.7117 22.7117 22.7117 22.7117 22.7117 22.7117 22.7117 22.7117
  22.7117 22.7117 22.7117 22.7117 22.7117 22.7117 22.7117 22.7117
  22.7117 22.7117]
```

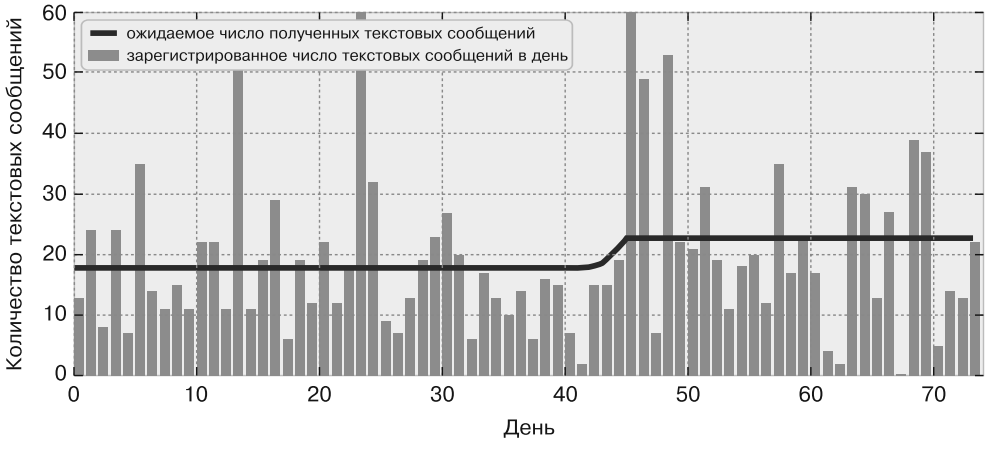

Рис. 1.7. Число полученных текстовых сообщений по сравнению с ожидаемым

Приведенные на рис. 1.7 результаты уверенно демонстрируют влияние точки ветвления. Впрочем, к этим результатам следует относиться с осторожностью; в линии, отражающей «ожидаемое число полученных сообщений», отсутствует неопределенность, а нам хотелось бы ее видеть. Наш анализ дает серьезные основания полагать, что поведение пользователя менялось (если бы это было не так, то значение λ_1 было бы ближе к λ_2), причем изменение было резким, а не постепенным (что демонстрирует выраженный пик апостериорного распределения τ). Можно только догадываться, что привело к этому изменению: понижение стоимости текстовых сообщений, подписка на прогноз погоды в текстовых сообщениях или новые романтические отношения.

1.5. Выводы

В этой главе мы поговорили о различиях между мировоззрением сторонников частотного подхода и байесовского толкования вероятностей. Вы познакомились с двумя важными распределениями вероятностей: пуассоновским и экспоненциальным. Этими «кирпичиками» мы воспользуемся для создания новых байесовских моделей, подобно вышеприведенному примеру с обменом текстовыми сообщениями. В главе 2 мы рассмотрим еще несколько схем моделирования и применения PyMC.

1.6. Приложение

1.6.1. Статистическое определение фактического различия двух параметров λ

В примере с обменом текстовыми сообщениями мы объявили параметры λ_1 и λ_2 различными на основе визуального изучения их апостериорных распределений. Этого было достаточно в силу визуальной отдаленности. А если бы это было не так? Что, если бы распределения частично пересекались? Как формализовать принятие этого решения?

Одно из возможных решений: вычислить вероятность $P(\lambda_1 < \lambda_2 \,|\, \text{данные})$, то есть определить, какова вероятность того, что при наблюдаемых данных фактическое значение λ_1 меньше, чем λ_2. Если это число близко к 50 % (что не лучше гадания путем подбрасывания монеты), то уверенности в том, что они действительно различаются, нет. Если же оно близко к 100 %, то наша уверенность в сильном различии двух фактических значений очень сильна. Вычислить этот показатель с помощью

выборок из апостериорных распределений очень просто — достаточно определить количество случаев, когда выборка из апостериорного распределения λ_1 меньше, чем для λ_2:

```
print lambda_1_samples < lambda_2_samples
# Булев массив: True, если lambda_1 меньше lambda_2.
```

```
[Output]:

[ True True True True ..., True True True True]
```

```
# Насколько часто это имеет место?
print (lambda_1_samples < lambda_2_samples).sum()

# Каково число выборок?
print lambda_1_samples.shape[0]

[Output]:

29994
30000

# Полученное отношение равно искомой вероятности.
# Или можно просто воспользоваться методом mean():
print (lambda_1_samples < lambda_2_samples).mean()
```

```
[Output]:

0.9998
```

Итак, вероятность равна почти 100 %, и мы можем быть твердо уверены в различии этих двух значений.

Можно задаться и более сложными вопросами, например: «Какова вероятность того, что значения различаются по крайней мере на 1? На 2? На 5? На 10?»

```
# Вектор abs(lambda_1_samples - lambda_2_samples) > 1 состоит
# из булевых значений, True — если значения различаются
# по крайней мере на 1, False — в противном случае.
# Насколько часто это имеет место? Воспользуемся .mean()
for d in [1,2,5,10]:
    v = (abs(lambda_1_samples - lambda_2_samples) >= d).mean()
    print "Какова вероятность того, что значения различаются более \
          чем на %d? %.2f"%(d,v)
```

```
[Output]:

Какова вероятность того, что значения различаются более чем на 1? 1.00
Какова вероятность того, что значения различаются более чем на 2? 1.00
Какова вероятность того, что значения различаются более чем на 5? 0.49
Какова вероятность того, что значения различаются более чем на 10? 0.00
```

1.6.2. Обобщаем на случай двух точек ветвления

Возможно, вам интересно, как можно обобщить предыдущую модель на случай нескольких точек ветвления или, возможно, поставить под вопрос наличие только одной точки ветвления. Начнем с обобщения модели на две точки ветвления (что влечет за собой наличие трех параметров λ_i). Модель выглядит очень похожей на предыдущую:

$$\lambda = \begin{cases} \lambda_1, \text{ если } t < \tau_1 \\ \lambda_2, \text{ если } \tau_1 \le t < \tau_2 \\ \lambda_3, \text{ если } t \ge \tau_2 \end{cases},$$

где

$$\lambda_1 \sim \text{Exp}(\alpha),$$
$$\lambda_2 \sim \text{Exp}(\alpha),$$
$$\lambda_3 \sim \text{Exp}(\alpha)$$

и

$$\tau_1 \sim \text{DiscreteUniform}(1, 69),$$
$$\tau_2 \sim \text{DiscreteUniform}(\tau_1, 70).$$

Давайте запрограммируем эту модель. Код тоже очень похож на предыдущий:

```
lambda_1 = pm.Exponential("lambda_1", alpha)
lambda_2 = pm.Exponential("lambda_2", alpha)
lambda_3 = pm.Exponential("lambda_3", alpha)

tau_1 = pm.DiscreteUniform("tau_1", lower=0, upper=n_count_data-1)
tau_2 = pm.DiscreteUniform("tau_2", lower=tau_1, upper=n_count_data)

@pm.deterministic
def lambda_(tau_1=tau_1, tau_2=tau_2,
            lambda_1=lambda_1, lambda_2=lambda_2, lambda_3 = lambda_3):
    out = np.zeros(n_count_data) # Количество точек ветвления
    out[:tau_1] = lambda_1       # lambda до точки tau равна lambda_1
    out[tau_1:tau_2] = lambda_2
    out[tau_2:] = lambda_3       # lambda после (и включая) точку tau
                                 # равна lambda_2
    return out
observation = pm.Poisson("obs", lambda_, value=count_data, observed=True)
model = pm.Model([observation, lambda_1, lambda_2, lambda_3, tau_1, tau_2])
mcmc = pm.MCMC(model)
mcmc.sample(40000, 10000)
```

```
[Output]:

[-----------------100%-----------------] 40000 of 40000 complete in 19.5 sec
```

На рис. 1.8 показаны апостериорные распределения пяти неизвестных величин. Как видим, модель обнаружила точки ветвления в дни 45-й и 47-й. Как вы думаете, не переобучилась ли модель на наших данных?

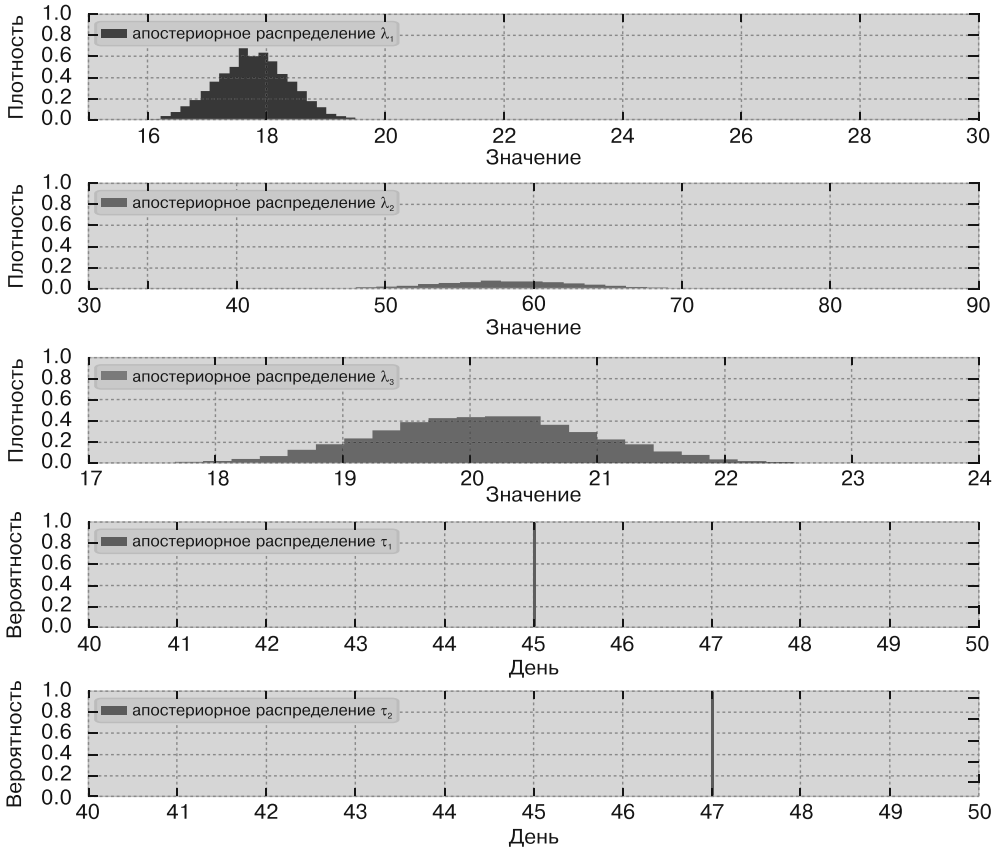

Рис. 1.8. Апостериорные распределения пяти неизвестных параметров в обобщенной модели текстовых сообщений

Конечно, у нас могут быть предположения о количестве точек ветвления в данных. Например, мне кажется, что одна точка ветвления вероятнее, чем две, что, в свою очередь, вероятнее, чем три, и т. д. Это значит, что нам нужно создать *априорное* распределение возможного количества точек ветвления — и пусть модель принимает решение! После обучения модель может решить, что да, вероятнее всего, здесь одна точка ветвления. Соответствующий код выходит за рамки данной главы, я лишь хотел обратить внимание, что к модели следует относиться с той же долей здорового скепсиса, что и к данным.

1.7. Упражнения

1. Каковы при заданных `lambda_1_samples` и `lambda_2_samples` средние значения апостериорных распределений λ_1 и λ_2?
2. Чему равно ожидаемое повышение (в процентах) частоты обмена сообщениями? **Подсказка:** вычислите среднее значение (`lambda_2_samples` – `lambda_1_samples`) / `lambda_1_samples`. Отмечу, что этот показатель существенно отличается от (`lambda_2_samples.mean()` – `lambda_1_samples.mean()`) / `lambda_1_samples.mean()`.
3. Чему равно среднее значение λ_1, если известно, что τ меньше 45? То есть предположим, что у нас появилась новая информация о том, что изменение поведения произошло ранее 45-го дня. Чему равно математическое ожидание λ_1 в этом случае? (Нет необходимости повторять все действия с PyMC. Достаточно учесть все экземпляры с `tau_samples` < 45.)

1.7.1. Ответы

1. Для вычисления средних значений апостериорных распределений (равных их математическим ожиданиям) нам достаточно выборок и функции `.mean`.

```
print lambda_1_samples.mean()
print lambda_2_samples.mean()
```

2. При заданных двух числах *a* и *b* относительный рост определяется по формуле $(a - b) / b$. В нашем случае мы не знаем точно, каковы значения λ_1 и λ_2. При вычислении:

```
(lambda_2_samples-lambda_1_samples)/lambda_1_samples
```

мы получаем в качестве результата еще один вектор, отражающий *апостериорное распределение относительного роста*. Например, как на рис. 1.9.

```
relative_increase_samples = (lambda_2_samples-lambda_1_samples)
                             /lambda_1_samples
print relative_increase_samples
```

```
[Output]:

[ 0.263 0.263 0.263 0.263 ..., 0.1622 0.1898 0.1883 0.1883]
```

```
figsize(12.5,4)
plt.hist(relative_increase_samples, histtype='stepfilled',
         bins=30, alpha=0.85, color="#7A68A6", normed=True,
         label='апостериорное распределение относительного роста')
```

```
plt.xlabel(u"Относительный рост")
plt.ylabel(u"Плотность относительного роста")
plt.title(u"Апостериорное распределение относительного роста")
plt.legend();
```

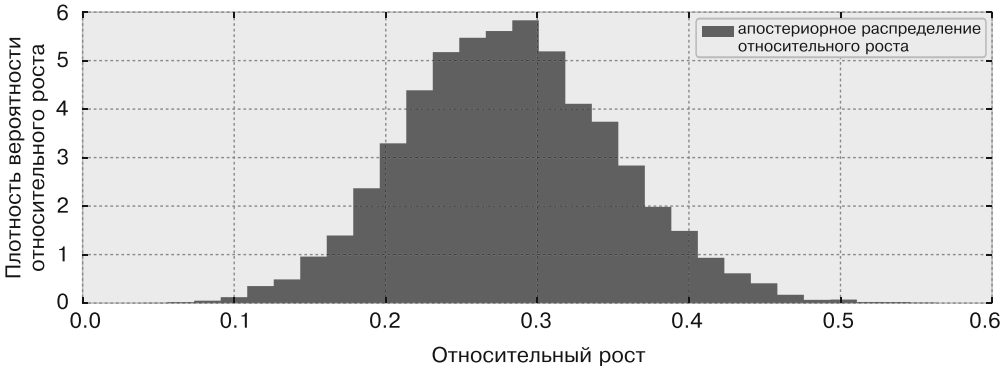

Рис. 1.9. Апостериорное распределение относительного роста

Для вычисления среднего значения достаточно получить среднее значение этого нового вектора:

```
print relative_increase_samples.mean()
```

```
[Output]:

0.280845247899
```

3. Если нам известно, что $\tau < 45$, то необходимо учесть только выборки, соответствующие этому условию:

```
ix = tau_samples < 45
print lambda_1_samples[ix].mean()
```

```
[Output]:

17.7484086925
```

1.8. Библиография

1. *Gelman A.* N Is Never Large. Statistical Modeling, Causal Inference and Social Science. http://andrewgelman.com/2005/07/31/n_is_never_larg/.
2. *Halevy A., Norvig P., Pereira F.* The Unreasonable Effectiveness of Data. IEEE Intelligent Systems, 24, № 2 (March/April 2009): 8–12.
3. *Lin J., Kolcz A.* Large-Scale Machine Learning at Twitter // Proceedings of the 2012 ACM SIGMOD International Conference on Management of Data (Scottsdale, AZ: May 2012), 793–804.
4. *Канеман Д.* Думай медленно... Решай быстро. — М.: АСТ, 2014.
5. *Anand P., Huard D., Fonnesbeck C. J.* PyMC: Bayesian Stochastic Modelling in Python // Journal of Statistical Software, 35, № 4 (2010): 1–81.
6. *Cronin B.* Why Probabilistic Programming Matters[1]. https://plus.google.com/u/0/107971134877020469960/posts/KpeRdJKR6Z1.

[1] Сервис Google+ прекратил свою работу 2 апреля 2019 года для обычных аккаунтов Google, поэтому ссылка на эту статью более недоступна. Вы можете прочесть другую статью этого же автора по данной теме по адресу https://www.oreilly.com/ideas/probabilistic-programming.

2 Еще немного о PyMC

2.1. Введение

В этой главе мы продолжим знакомство с синтаксисом и паттернами проектирования PyMC, а также поговорим о моделировании системы с байесовской точки зрения. Здесь вы найдете описание способов визуализации данных для определения критерия согласия байесовской модели, а также советы по их использованию.

2.1.1. Связи «предок — потомок»

Введем для удобства описания байесовских связей и ради согласованности с документацией PyMC понятия *переменной-предка* (parent variable) и *переменной-потомка* (child variable).

- *Переменные-предки* — переменные, которые влияют на другие переменные.
- *Переменные-потомки* — переменные, на которые влияют другие переменные, то есть зависимые от переменных-предков.

Переменная может быть одновременно и предком, и потомком. Например, рассмотрим следующий код PyMC.

```
import pymc as pm

lambda_ = pm.Exponential("poisson_param", 1)
# Используется при обращении к следующей переменной...
data_generator = pm.Poisson("data_generator", lambda_)

data_plus_one = data_generator + 1
```

Переменная `lambda_` представляет собой параметр переменной `data_generator`, а потому влияет на ее значения. Первая является предком последней. И наоборот, `data_generator` — потомок `lambda_`.

Аналогично `data_generator` — предок переменной `data_plus_one` (что делает `data_generator` одновременно и предком, и потомком (для разных переменных)). `data_plus_one` следует рассматривать как переменную PyMC, хотя она на таковую и не похожа, поскольку является *функцией* от другой переменной PyMC, а значит, переменной-потомком по отношению к `data_generator`.

Данная терминология призвана помочь в описании связей при моделировании PyMC. Обращаться к переменным-предкам и переменным-потомкам заданной переменной можно с помощью ее атрибутов `children` и `parents`.

```
print "Потомки 'lambda_': "
print lambda_.children
print "\nПредки 'data_generator': "
print data_generator.parents
print "\nПотомки 'data_generator': "
print data_generator.children
```

```
[Output]:

Потомки 'lambda_':
set([<pymc.distributions.Poisson 'data generator' at 0x10e093490>])

Предки 'data generator':
{'mu': <pymc.distributions.Exponential 'poisson param' at 0x10e093610>}

Потомки 'data generator':
set([<pymc.PyMCObjects.Deterministic '(data generator add 1)'
    at 0x10e093150>])
```

Конечно, у потомка может быть несколько предков, а у предка может быть один или несколько потомков.

2.1.2. Переменные PyMC

Для всех переменных PyMC доступно свойство `value`. Оно содержит *текущее* (возможно, случайное) внутреннее значение переменной. Если переменная — потомок, то ее значение меняется при изменении значений родительских переменных. На примере тех переменных, что и выше:

```
print "lambda_.value =", lambda_.value
print "data_generator.value =", data_generator.value
print "data_plus_one.value =", data_plus_one.value
```

```
[Output]:

lambda .value = 1.0354800596
data generator.value = 4
data plus one.value = 5
```

Для PyMC важны два типа переменных: *детерминистические* (класс `Deterministic`) и *стохастические* (класс `Stochastic`).

- **Детерминистическая переменная** — такая, значение которой не случайно, а четко определено значениями ее переменных-предков. Чтобы не запутаться, можно провести быструю мысленную проверку: *если я знаю значения всех переменных-предков переменной* `foo`*, то могу точно определить значение* `foo`.
- **Стохастическая переменная** — недетерминистическая, то есть такая, что даже при известных значениях всех ее переменных-предков (если у нее вообще есть предки) ее значение все равно может оказаться произвольным. К этой категории можно отнести экземпляры классов `Poisson`, `DiscreteUniform` и `Exponential`.

Инициализация стохастических переменных

Первый аргумент при инициализации стохастической переменной — строка с именем этой переменной. За ней следуют дополнительные аргументы, зависящие от класса. Например:

```
some_variable = pm.DiscreteUniform("discrete_uni_var",0,4)
```

где 0 и 4 — нижняя и верхняя границы случайной переменной, специфические параметры для класса `DiscreteUniform`. В документации PyMC (http://pymc-devs.github.com/pymc/distributions.html) приведен список соответствующих параметров для стохастических переменных (или же можете воспользоваться командой `??`, если вы работаете с IPython!).

Аргументом-именем при дальнейшем анализе можно воспользоваться для извлечения апостериорного распределения, так что желательно, чтобы это имя было информативным. Обычно я применяю с этой целью название переменной Python.

Для задач с несколькими переменными вместо создания массива Python стохастических переменных лучше указать при обращении к стохастической переменной ключевое слово `size`, в результате чего будет создан массив (независимых) стохастических переменных. Этот массив ведет себя аналогично массиву NumPy, и ссылка на его атрибут `value` возвращает массив NumPy.

Аргумент `size` также решает неприятную проблему с необходимостью моделирования большого количества случайных переменных β_i, $i = 1, ..., N$. Вместо произвольных названий и переменных для каждой из них вот так:

```
beta_1 = pm.Uniform("beta_1", 0, 1)
beta_2 = pm.Uniform("beta_2", 0, 1)
...
```

можно обернуть их в одну переменную:

```
betas = pm.Uniform("betas", 0, 1, size=N)
```

Вызов метода random()

Можно также вызвать метод `random()` стохастической переменной, который (по заданным значениям-предкам) возвращает новое, случайное значение. Продемонстрируем это на примере с обменом текстовыми сообщениями из главы 1.

```
lambda_1 = pm.Exponential("lambda_1", 1)
  # априорное распределение для первого варианта поведения
lambda_2 = pm.Exponential("lambda_2", 1)
  # априорное распределение для второго варианта поведения
tau = pm.DiscreteUniform("tau", lower=0, upper=10)
  # априорное распределение при смене поведения

print "Инициализированные значения..."
print "lambda_1.value: %.3f" % lambda_1.value
print "lambda_2.value: %.3f" % lambda_2.value
print "tau.value: %.3f" % tau.value
print lambda_1.random(), lambda_2.random(), tau.random()

print "После вызова метода random() для этих переменных..."
print "lambda_1.value: %.3f" % lambda_1.value
print "lambda_2.value: %.3f" % lambda_2.value
print "tau.value: %.3f" % tau.value
```

```
[Output]:

Инициализированные значения...
lambda 1.value: 0.813
lambda 2.value: 0.246
tau.value: 10.000

После вызова метода random() для этих переменных...
lambda 1.value: 2.029
lambda 2.value: 0.211
tau.value: 4.000
```

Вызов метода `random()` приводит к сохранению нового значения в атрибуте `value` переменной.

Детерминистические переменные

Поскольку большинство моделируемых нами переменных будут стохастическими, мы будем выделять детерминистические переменные с помощью адаптера `pymc.deterministic` (если вы еще не сталкивались с адаптерами Python, которые также называются *декораторами*, не проблема. Просто вставьте декоратор `pymc.deterministic` перед объявлением переменной, и можно работать дальше). Детерминистическая переменная объявляется в виде функции языка Python:

```
@pm.deterministic
def some_deterministic_var(v1=v1,):
    #код...
```

Фактически `some_deterministic_var` можно рассматривать как переменную, а не функцию языка Python.

Добавление перед объявлением адаптера — простейший, но не единственный способ создания детерминистических переменных. Элементарные операции, к примеру сложение, возведение в степень и т. п., приводят к неявному созданию детерминистических переменных. Например, следующее выражение возвращает детерминистическую переменную:

```
type(lambda_1 + lambda_2)
```

```
[Output]:

pymc.PyMCObjects.Deterministic
```

Мы уже сталкивались с использованием адаптера `deterministic` в примере с обменом текстовыми сообщениями в главе 1. Тогда модель для λ выглядела так:

$$\lambda = \begin{cases} \lambda_1, \text{ если } t < \tau \\ \lambda_2, \text{ если } t \geq \tau \end{cases}.$$

В виде кода PyMC:

```
import numpy as np
n_data_points = 5 # в главе 1 было ~70 точек данных

@pm.deterministic
def lambda_(tau=tau, lambda_1=lambda_1, lambda_2=lambda_2):
    out = np.zeros(n_data_points)
    out[:tau] = lambda_1 # lambda до точки tau равна 1
    out[tau:] = lambda_2 # lambda после точки tau равна 2
    return out
```

Очевидно, что при известных τ, λ_1 и λ_2 значение переменной λ тоже всегда известно; следовательно, она является детерминистической.

Внутри декоратора `deterministic` передаваемые стохастические переменные ведут себя как скалярные значения или массивы NumPy (в случае нескольких переменных), а *не* как стохастические переменные. Например, при выполнении следующего кода:

```
@pm.deterministic
def some_deterministic(stoch=some_stochastic_var):
return stoch.value**2
```

будет возвращена ошибка `AttributeError` с уточнением, что у переменной `stoch` отсутствует атрибут `value`. Во время этапа обучения всякий раз передается `value` стохастической переменной, а не она сама.

Обратите внимание, что при создании детерминистической функции мы применяли именованные аргументы (keyword arguments) для каждой используемой в функции переменной. Это обязательный шаг — все переменные должны задаваться с помощью именованных аргументов.

2.1.3. Учет наблюдений в модели

На текущий момент, хотя это и не очевидно, наши априорные распределения полностью определены. Например, можно ставить вопросы вроде «Каково априорное распределение λ_1?» и успешно отвечать на них (рис. 2.1).

```
%matplotlib inline
from IPython.core.pylabtools import figsize
from matplotlib import pyplot as plt
figsize(12.5, 4)
plt.rcParams['savefig.dpi'] = 300
plt.rcParams['figure.dpi'] = 300

samples = [lambda_1.random() for i in range(20000)]
plt.hist(samples, bins=70, normed=True, histtype="stepfilled")
plt.title("Априорное распределение $\lambda_1$")
plt.xlabel(u"Значение")
plt.ylabel(u"Плотность распределения")
plt.xlim(0, 8);
```

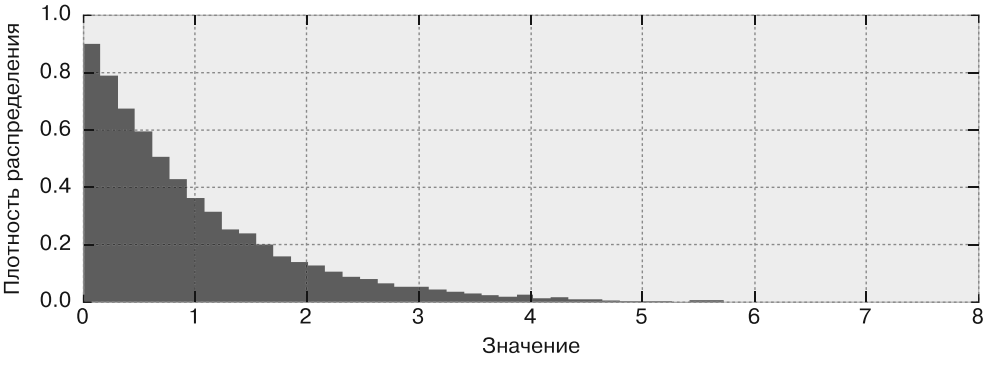

Рис. 2.1. Априорное распределение λ_1

Если говорить в терминологии главы 1 (хотя это будет и не совсем правильно), мы задали $P(A)$. Наша следующая задача — учесть данные/факты/наблюдения X в нашей модели. Сейчас мы этим и займемся.

У стохастических переменных PyMC есть дополнительный именованный аргумент `observed`, принимающий булево значение (по умолчанию `False`). Роль ключевого слова

observed очень проста: зафиксировать текущее значение переменной, то есть сделать value неизменяемым. При этом при создании переменной необходимо задать начальное value, соответствующее наблюдениям, которые мы хотим учесть, обычно в виде массива (и ради повышения производительности — массива NumPy). Например:

```
data = np.array([10, 5])
fixed_variable = pm.Poisson("fxd", 1, value=data, observed=True)
print "значение: ", fixed_variable.value
print "вызываем .random()"
fixed_variable.random()
print "значение: ", fixed_variable.value
```

```
[Output]:

значение: [10 5]
вызываем .random()
значение: [10 5]
```

Именно путем инициализации стохастической переменной с *фиксированным значением* и учитываются данные в моделях.

В завершение нашего примера с обменом текстовыми сообщениями привяжем переменную PyMC observations к набору полученных при наблюдении данных.

```
# Воспользуемся фиктивными данными
data = np.array([10, 25, 15, 20, 35])
obs = pm.Poisson("obs", lambda_, value=data, observed=True)
print obs.value
```

```
[Output]:

[10 25 15 20 35]
```

2.1.4. И наконец...

Мы обернули все созданные переменные в класс pm.Model. Благодаря ему можно анализировать переменные как единое целое. Этот шаг необязательный, поскольку вместо объекта класса Model можно передать в алгоритм обучения массив переменных. В последующих примерах мы иногда будем использовать этот шаг, а иногда — нет.

```
model = pm.Model([obs, lambda_, lambda_1, lambda_2, tau])
```

Результаты обучения модели можно найти в подразделе 1.4.1.

2.2. Подходы к моделированию

Перед тем как приступить к байесовскому моделированию, неплохо бы задуматься, *каким образом могли быть сгенерированы данные*. Поставьте себя на место эдакого всеведущего наблюдателя и попытайтесь представить, как бы *вы* воссоздали этот набор данных.

В главе 1 мы изучали данные, полученные в результате обмена текстовыми сообщениями. И начали с того, что задали себе вопрос: а как могли быть сгенерированы наши наблюдения?

1. Мы начали с размышлений: «Какая случайная переменная лучше всего подходит для описания этих данных?» Хороший кандидат на эту роль — пуассоновская случайная переменная, поскольку она может хорошо отражать подобные данные. Поэтому мы смоделировали число полученных текстовых сообщений на основе пуассоновского распределения.
2. Далее мы задумались: «Хорошо. Если допустить, что число текстовых сообщений распределено по закону Пуассона, то что нам нужно для пуассоновского распределения?» Что ж, у пуассоновского распределения есть параметр λ.
3. Известен ли нам параметр λ? Нет. На самом деле мы даже подозреваем, что значений λ может быть два, одно — для первого варианта поведения, а второе — для последующего поведения. Мы не знаем, когда происходит смена поведения, но назовем точку ветвления τ.
4. Какое распределение хорошо подходит для двух λ? Неплохо подходит экспоненциальное — оно задает вероятности положительных вещественных чисел. Что ж, у экспоненциального распределения тоже есть параметр — назовем его α.
5. Известно ли нам возможное значение параметра α? Нет. Мы можем продолжить и выбрать распределение для α, но лучше будет остановиться по достижении заданного уровня неведения. Хотя относительно λ у нас есть определенная априорная уверенность («Вероятно, λ меняется с течением времени», «Вероятно, значение λ — от 10 до 30» и т. д.), никакой уверенности относительно α у нас нет. Так что лучше на этом прекратить моделирование.

   Какое значение в таком случае хорошо подойдет для α? Мы предполагаем, что значения параметров λ находятся между 10 и 30, так что очень низкое значение α (соответствующее большей вероятности высоких значений) плохо отражает нашу априорную уверенность. Точно так же нашу априорную уверенность плохо отражает слишком высокое значение α. Все это было продемонстрировано в главе 1.
6. У нас нет экспертного мнения о возможном дне τ. Так что предположим, что величина τ равномерно дискретно распределена по всему промежутку времени.

На рис. 2.2 приведена наглядная визуализация вышеописанного, на которой стрелки соответствуют отношениям «предок — потомок» (получено с помощью библиотеки Daft языка Python, http://daft-pgm.org/).

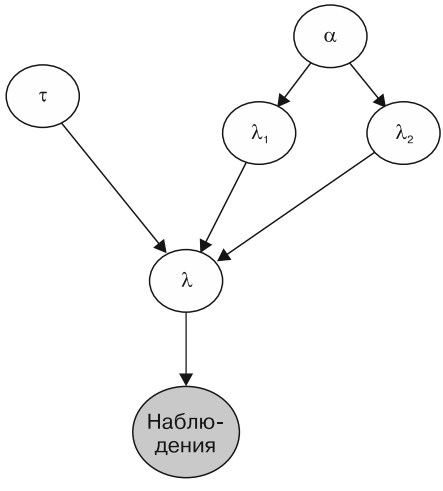

Рис. 2.2. Наглядная модель схемы генерации наблюдений

PyMC, как и другие вероятностные языки программирования, был изначально предназначен для изложения подобных *историй* генерации данных. Кронин пишет [1]:

> «Вероятностное программирование раскрывает историю возникновения данных — святая святых бизнес-аналитики и невоспетую героиню научных доказательств. Человеческое мышление оперирует историями, поэтому столь неоправданно велика роль различных притч в принятии решений, обоснованных или нет. Но описать такую историю существующая аналитика обычно не способна; вместо этого числа как будто берутся с потолка, практически без причинно-следственного контекста, столь важного для принятия людьми решений».

2.2.1. Та же история, но с другой концовкой

Что интересно, путем пересказа историй можно создавать *новые наборы данных*. Например, если пройти по шести вышеописанным шагам в обратном направлении, можно смоделировать вероятную реализацию нашего набора данных.

В дальнейшем мы воспользуемся внутренними функциями PyMC для генерации случайных (но не стохастических) переменных. Функция `rdiscrete_uniform` служит для генерации случайных результатов, взятых из равномерного дискретного распределения (аналогично методу `numpy.random.randint`).

1. Указываем момент смены характера поведения пользователя с помощью выборки из DiscreteUniform(0, 80)[1].

```
tau = pm.rdiscrete_uniform(0, 80)
print tau
```

```
[Output]:

29
```

2. Берем λ_1 и λ_2 из распределения Exp(α)[2]:

```
alpha = 1./20.
lambda_1, lambda_2 = pm.rexponential(alpha, 2)
print lambda_1, lambda_2
```

```
[Output]:

27.5189090326 6.54046888135
```

3. Для предшествующих τ дней $\lambda = \lambda_1$; для следующих за τ дней $\lambda = \lambda_2$.

```
lambda_ = np.r_[ lambda_1*np.ones(tau), lambda_2*np.ones(80-tau) ]
print lambda_
```

```
[Output]:

[ 27.519 27.519 27.519 27.519 27.519 27.519 27.519 27.519 27.519
  27.519 27.519 27.519 27.519 27.519 27.519 27.519 27.519 27.519
  27.519 27.519 27.519 27.519 27.519 27.519 27.519 27.519 27.519
  27.519 27.519  6.54   6.54   6.54   6.54   6.54   6.54   6.54
   6.54   6.54   6.54   6.54   6.54   6.54   6.54   6.54   6.54
   6.54   6.54   6.54   6.54   6.54   6.54   6.54   6.54   6.54
   6.54   6.54   6.54   6.54   6.54   6.54   6.54   6.54   6.54
   6.54   6.54   6.54   6.54   6.54   6.54   6.54   6.54 ]
```

4. Производим выборку из Poi(λ_1)[3] для дней, предшествующих τ, и Poi(λ_2) для следующих за τ дней. Например:

```
data = pm.rpoisson(lambda_)
print data
```

```
[Output]:

[36 22 28 23 25 18 30 27 34 26 33 31 26 26 32 26 23 32 33 33 27 26 35 20 32
```

---

[1] Равномерное дискретное распределение в интервале (0, 80).

[2] Экспоненциальное распределение с параметром α.

[3] Пуассоновское распределение с параметром λ_1.

```
 44 23 30 26  9 11  9  6  8  7  1  8  5  6  5  9  5  7  6  5 11  5  5 10  9
  4  5  7  5  9  8 10  5  7  9  5  6  3  8  7  4  6  7  7  4  5  3  5  6  8
 10  5  6  8  5]
```

5. Строим график искусственно созданного набора данных (рис. 2.3):

```
plt.bar(np.arange(80), data, color="#348ABD")
plt.bar(tau-1, data[tau - 1], color="r", label=u"изменение поведения пользователя")
plt.xlabel(u"Время (дней)")
plt.ylabel(u"Получено текстовых сообщений")
plt.title(u"Искусственный набор данных, полученный путем\
          имитационного моделирования")
plt.xlim(0, 80)
plt.legend();
```

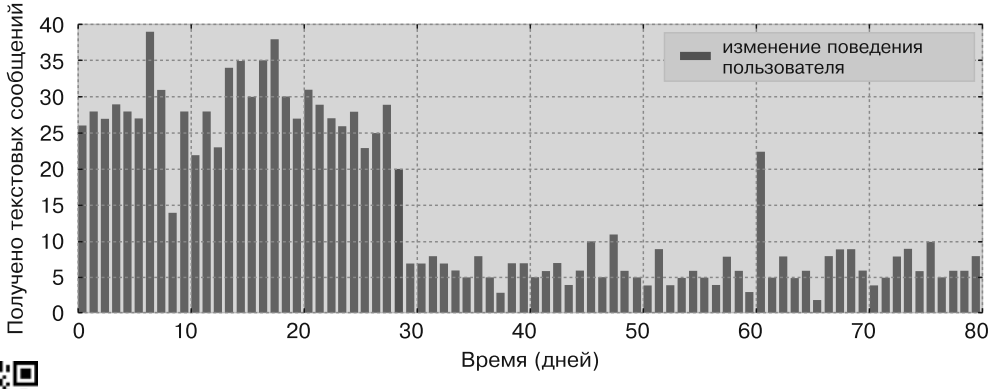

Рис. 2.3. Искусственный набор данных, полученный путем имитационного моделирования

Ничего удивительного, что наш фиктивный набор данных с рис. 2.3 не похож на данные наблюдений; более того, вероятность подобного сходства невероятно мала. Движок PyMC ориентирован на поиск хороших значений параметров λ_i, τ, которые бы максимизировали эту вероятность.

Возможность генерации искусственных наборов данных — интересный побочный эффект моделирования, и, как мы увидим, она представляет собой важный метод байесовского вывода. Например, сгенерируем еще несколько наборов данных (рис. 2.4).

```
def plot_artificial_sms_dataset():
    tau = pm.rdiscrete_uniform(0, 80)
    alpha = 1./20.
    lambda_1, lambda_2 = pm.rexponential(alpha, 2)
    data = np.r_[pm.rpoisson(lambda_1, tau), pm.rpoisson(lambda_2, 80 - tau)]
    plt.bar(np.arange(80), data, color="#348ABD")
    plt.bar(tau - 1, data[tau-1], color="r",
```

```
    label=u"изменение поведения пользователя")
    plt.xlim(0, 80)
    plt.xlabel(u"Время (дней)")
    plt.ylabel(u"Получено текстовых сообщений")

figsize(12.5, 5)
plt.title(u"Еще несколько искусственных наборов данных, полученных\
          путем имитационного моделирования")
for i in range(4):
    plt.subplot(4, 1, i+1)
    plt.xlabel(u"Время (дней)")
    plt.ylabel(u"Получено текстовых сообщений")
    plot_artificial_sms_dataset()
```

Далее будет показано, как на основе этого выполнять предсказания и проверять пригодность моделей.

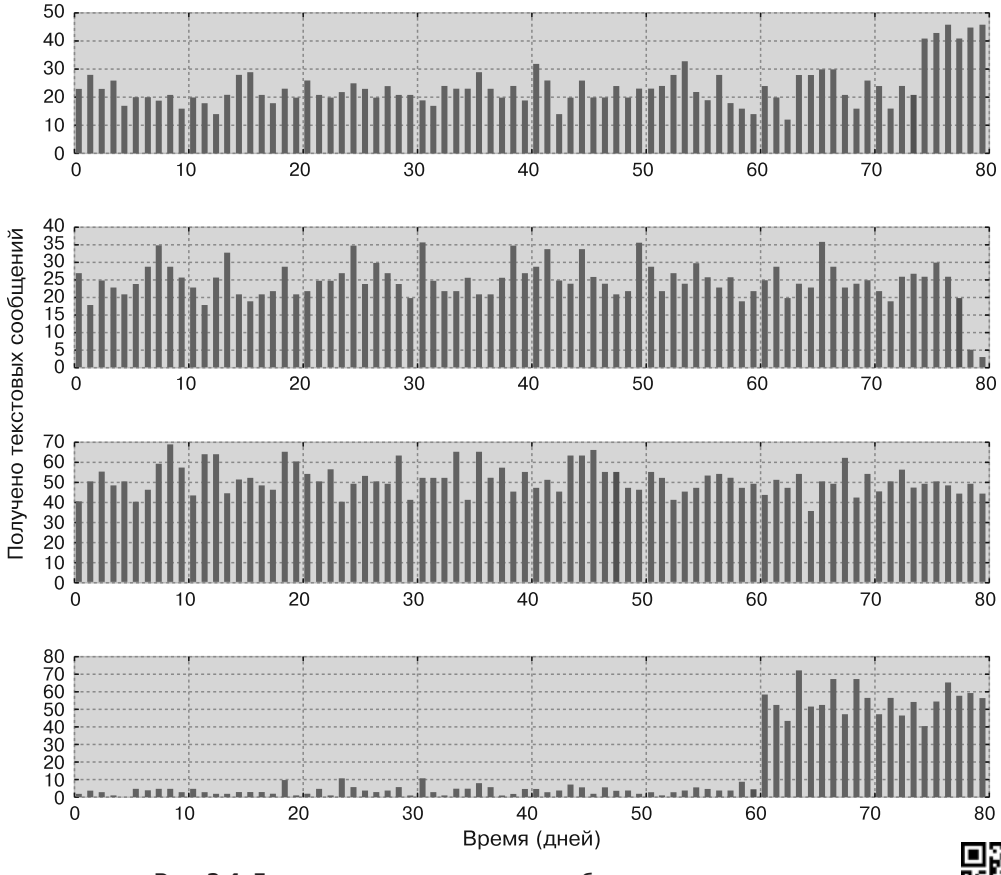

Рис. 2.4. Еще несколько искусственных наборов данных, полученных путем имитационного моделирования

2.2.2. Пример: байесовское A/B-тестирование

A/B-тестирование — статистический паттерн проектирования для определения различий эффективности двух разных методов лечения. Например, фармацевтической компании требуется сравнить эффективность лекарства A с эффективностью лекарства B. Компания тестирует лекарство A на одной части группы пациентов, а лекарство B — на остальных (обычно группа разбивается пополам, 50/50, но мы снимем это ограничение). После достаточного количества клинических испытаний статистики компании оценивают эффективность, чтобы решить, какое лекарство демонстрирует лучшие результаты.

Аналогично веб-разработчиков клиентской части интересует, при каком дизайне их сайта количество *конверсий*[1] (conversion) будет выше, причем конверсией может быть регистрация пользователя, покупка чего-либо или какое-то другое действие. Для этого часть пользователей перенаправляется на сайт A, а другая часть — на сайт B (с другим дизайном). В то же время фиксируется, привело ли посещение сайта к конверсии. Вся эта информация в дальнейшем также фиксируется и анализируется.

Главное в A/B-тестах то, что между группами есть только одно различие. Следовательно, любое существенное изменение метрик (эффективности лекарств или количества конверсий) можно приписать непосредственно этому различию.

Зачастую анализ после эксперимента производится с помощью так называемой проверки статистической гипотезы, например *проверки различия средних значений* (difference of means test) или *проверки различия долей* (difference of proportions test). Это требует использования такого запутанного понятия, как «*Z*-оценка», и еще более запутанных малопонятных «*p*-значений» (даже не спрашивайте, что это такое). Если вы слушали курс статистики, то вас, возможно, обучали этой методике (хотя вовсе не факт, что вы в ней *разобрались*). И если мы с вами в этом схожи, то их вывод отнюдь не доставляет вам радости. Если так — отлично. Байесовский подход к решению этой задачи гораздо более естественен.

2.2.3. Простой случай

Поскольку книга предназначена для программистов, то мы возьмем за основу пример с разработкой сайтов. Пока мы сосредоточим наши усилия на анализе сайта A. Допустим, что существует вероятность, равная p_A, конверсии пользователей, которым продемонстрировали сайт A. Это показатель его фактической эффективности. Пока данная величина нам неизвестна.

Допустим, что сайт A показали N людям, из которых n совершили конверсию. Можно необдуманно решить, что $p_A = n / N$. К сожалению, *наблюдаемая часто-*

[1] См.: https://ru.wikipedia.org/wiki/Конверсия_(в_интернет-маркетинге).

та n / N вовсе не обязательно равна p_A; *наблюдаемая частота* (observed frequency) и *фактическая частота* (true frequency) наступления событий — разные вещи. Фактическая частота — это вероятность наступления события, и она отнюдь не всегда равна наблюдаемой частоте. Например, фактическая частота выпадения 1 на шестигранном кубике равна 1/6, но если бросить кубик шесть раз, то 1 может вовсе ни разу не выпасть (наблюдаемая частота)! В большинстве случаев приходится определять фактическую частоту событий как:

- долю пользователей, которые купили товары;
- долю в генеральной совокупности особей с какой-то характерной особенностью;
- процент интернет-пользователей, у которых есть кошки;
- вероятность того, что завтра пойдет дождь.

К сожалению, помехи и прочие сложности скрывают от нас фактическую частоту и ее приходится выводить из наблюдаемых данных. Для вывода вероятных значений фактической частоты мы воспользуемся байесовскими статистическими методами на основе соответствующих априорных распределений и наблюдаемых данных. Применительно к нашему примеру с конверсиями хотелось бы оценить возможное значение p_A — фактической частоты конверсий — с помощью имеющейся информации, а именно: N (общего количества посетителей) и n (числа конверсий).

Для создания байесовской модели необходимо задать априорные распределения для наших неизвестных величин. Как вы полагаете *априори*, какой может быть p_A? В данном примере у нас нет четкой уверенности относительно p_A, так что пока предположим, что p_A распределена равномерно по отрезку [0; 1]:

```
import pymc as pm

# Параметры представляют собой границы равномерного распределения.
p = pm.Uniform('p', lower=0, upper=1)
```

Предположим для этого примера, что $p_A = 0{,}05$, а сайт А был показан $N = 1500$ пользователям. Проведем имитационное моделирование того, совершил ли пользователь покупку. Для этого моделирования при N испытаниях мы воспользуемся **распределением Бернулли** (Bernoulli distribution). Оно представляет собой бинарную случайную переменную (которая может принимать только значения 0 или 1). Оно подходит для нашего случая, поскольку наши наблюдения также бинарны (имела место конверсия или нет). На более формальном языке: если $X \sim \text{Ber}(p)$, то $X = 1$ с вероятностью p и 0 с вероятностью $1 - p$. Конечно, на практике величина p_A нам неизвестна, но мы воспользуемся ею здесь для имитационного моделирования искусственных данных.

```
# Задаем ограничения
p_true = 0.05 # Помните, что на практике это значение неизвестно
```

```
N = 1500

# Выбираем N бернуллиевых случайных переменных из распределения Ber(0,05).
# Вероятность быть равной 1 для каждой из них составляет 0,05.
# Шаг генерации данных.
occurrences = pm.rbernoulli(p_true, N)

print occurrences # Помните: язык Python считает True == 1, а False == 0.
print occurrences.sum()
```

```
[Output]:

[False False False False ..., False False False False]
85
```

Наблюдаемая частота:

```
# Occurrences.mean() равна n/N.
print "Чему равна наблюдаемая частота в группе A? %.4f"\
                % occurrences.mean()
print "Равна ли наблюдаемая частота фактической? %s"\
                % (occurrences.mean() == p_true)
```

```
[Output]:

Чему равна наблюдаемая частота в группе A? 0.0567
Равна ли наблюдаемая частота фактической? False
```

Объединяем наблюдаемые данные в переменную PyMC с аргументом `observed = True` и запускаем алгоритм вывода:

```
# Учитываем данные, распределенные по закону Бернулли.
obs = pm.Bernoulli("obs", p, value=occurrences, observed=True)

# Будет объяснено в главе 3
mcmc = pm.MCMC([p, obs])
mcmc.sample(20000, 1000)
```

```
[Output]:

[-----------------100%-----------------] 20000 of 20000 complete in 2.0 sec
```

Строим график апостериорного распределения неизвестной величины p_A (рис. 2.5).

```
figsize(12.5, 4)
plt.title(u"Апостериорное распределение $p_A$, фактическая эффективность сайта A")
plt.vlines(p_true, 0, 90, linestyle="--", label=u"фактическая $p_A$ (неизвестна)")
plt.hist(mcmc.trace("p")[:], bins=35, histtype="stepfilled", normed=True)
```

```
plt.xlabel(u"Значение $p_A$")
plt.ylabel(u"Плотность")
plt.legend();
```

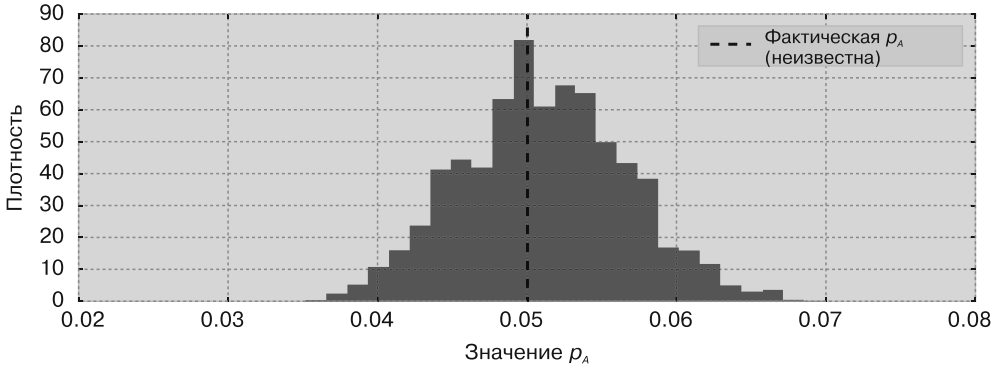

Рис. 2.5. Апостериорное распределение p_A, фактическая эффективность сайта A

Основная «масса» нашего апостериорного распределения сконцентрирована возле вероятного, судя по данным, фактического значения p_A: чем выше график распределения, тем вероятнее, что фактическое значение именно здесь. Попробуйте поменять число наблюдений N и посмотрите, как будет меняться апостериорное распределение.

Почему ось Y больше 1? Прекрасный ответ на этот вопрос можно найти на вебстранице [2] из списка литературы.

2.2.4. A и B вместе

Можно провести аналогичный анализ и для характеристик сайта B, чтобы узнать апостериорное распределение p_B. Выведем p_A, p_B и delta = $p_A - p_B$ одновременно. Это можно сделать с помощью детерминистических переменных PyMC. Предположим, что $p_B = 0{,}04$ (хотя мы этого не знаем), так что delta = 0,01, $N_B = 750$ (только половина от N_A), а данные сайта B мы получим путем имитационного моделирования аналогично тому, как мы сделали для сайта A.

```
import pymc as pm
figsize(12, 4)

# Эти две величины нам неизвестны.
true_p_A = 0.05
true_p_B = 0.04

# Обратите внимание, что неодинаковые размеры выборок не доставляют
```

```
# никаких сложностей при байесовском анализе.
N_A = 1500
N_B = 750

# Генерируем наблюдения.
observations_A = pm.rbernoulli(true_p_A, N_A)
observations_B = pm.rbernoulli(true_p_B, N_B)
print "Наблюдения с сайта A: ", observations_A[:30].astype(int), "..."
print "Наблюдения с сайта B: ", observations_B[:30].astype(int), "..."
```

```
[Output]:

Наблюдения с сайта A: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
           0 0 0 0 0 0 0] ...
Наблюдения с сайта B: [0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
           0 0 0 0 0 0 0] ...
```

```
print observations_A.mean()
print observations_B.mean()
```

```
[Output]:

0.0506666666667
0.0386666666667
```

```
# Создаем модель PyMC. Опять предполагаем, что априорные
# распределения p_A и p_B — равномерные.
p_A = pm.Uniform("p_A", 0, 1)
p_B = pm.Uniform("p_B", 0, 1)

# Описываем детерминистическую функцию delta. Это и есть интересующая нас
# неизвестная величина.
@pm.deterministic
def delta(p_A=p_A, p_B=p_B):
    return p_A - p_B

# Набор данных наблюдений; в этом случае у нас два набора данных наблюдений.
obs_A = pm.Bernoulli("obs_A", p_A, value=observations_A, observed=True)
obs_B = pm.Bernoulli("obs_B", p_B, value=observations_B, observed=True)

# будет объяснено в главе 3
mcmc = pm.MCMC([p_A, p_B, delta, obs_A, obs_B])
mcmc.sample(25000, 5000)
```

```
[Output]:

[-----------------100%-----------------] 25000 of 25000 complete
    in 3.8 sec
```

Построим графики апостериорных распределений для трех неизвестных величин (рис. 2.6).

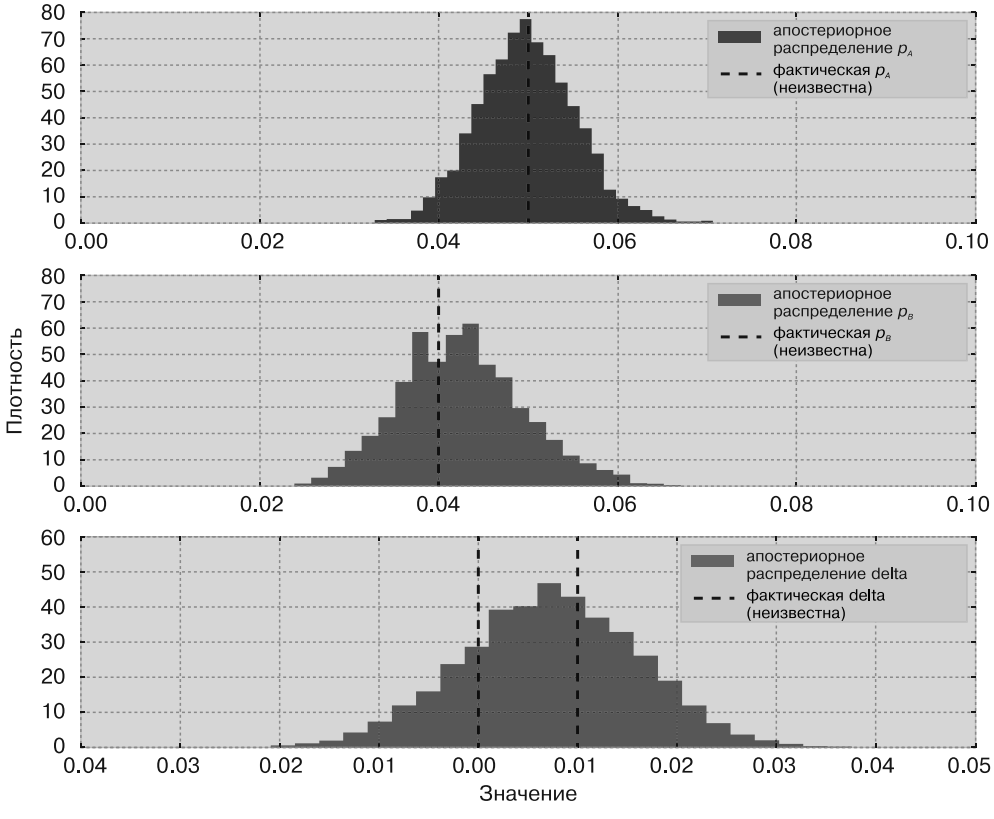

Рис. 2.6. Апостериорные распределения неизвестных величин p_A, p_B и delta

```
p_A_samples = mcmc.trace("p_A")[:]
p_B_samples = mcmc.trace("p_B")[:]
delta_samples = mcmc.trace("delta")[:]

figsize(12.5, 10)

# Гистограмма апостериорных распределений

ax = plt.subplot(311)

plt.xlim(0, .1)
plt.hist(p_A_samples, histtype='stepfilled', bins=30, alpha=0.85,
         label=u"апостериорное распределение $p_A$", color="#A60628",
         normed=True)
```

```
plt.vlines(true_p_A, 0, 80, linestyle="--",
           label=u"фактическая $p_A$ (неизвестна)")
plt.legend(loc="upper right")
plt.title(u"Апостериорные распределения неизвестных величин $p_A$, $p_B$ и\
          delta")
plt.ylim(0,80)

ax = plt.subplot(312)
plt.xlim(0, .1)
plt.hist(p_B_samples, histtype='stepfilled', bins=30, alpha=0.85,
         label=u"апостериорное распределение $p_B$", color="#467821",
         normed=True)
plt.vlines(true_p_B, 0, 80, linestyle="--",
           label=u"фактическая $p_B$ (неизвестна)")
plt.legend(loc="upper right")
plt.ylim(0,80)

ax = plt.subplot(313)
plt.hist(delta_samples, histtype='stepfilled', bins=30, alpha=0.85,
         label=u"апостериорное распределение delta", color="#7A68A6",
         normed=True)
plt.vlines(true_p_A - true_p_B, 0, 60, linestyle="--",
           label=u"фактическая delta (неизвестна)")
plt.vlines(0, 0, 60, color="black", alpha=0.2)
plt.xlabel(u"Значение")
plt.ylabel(u"Плотность")
plt.legend(loc="upper right");
```

Обратите внимание, что `N_B` < `N_A`, то есть данных с сайта B меньше, чем с сайта A, — апостериорное распределение p_B более размытое, а значит, наша уверенность в фактическом значении p_B слабее, чем в p_A. Чтобы это было легче заметить, построим два апостериорных распределения на одном графике (рис. 2.7):

```
figsize(12.5, 3)

# Гистограмма апостериорных распределений

plt.xlim(0, .1)
plt.hist(p_A_samples, histtype='stepfilled', bins=30, alpha=0.80,
         label=u"апостериорное распределение $p_A$", color="#A60628",
         normed=True)

plt.hist(p_B_samples, histtype='stepfilled', bins=30, alpha=0.80,
         label=u"апостериорное распределение $p_B$", color="#467821",
         normed=True)
plt.legend(loc="upper right")
plt.xlabel(u"Значение")
plt.ylabel(u"Плотность")
plt.title("Апостериорные распределения неизвестных $p_A$ и $p_B$")
plt.ylim(0,80);
```

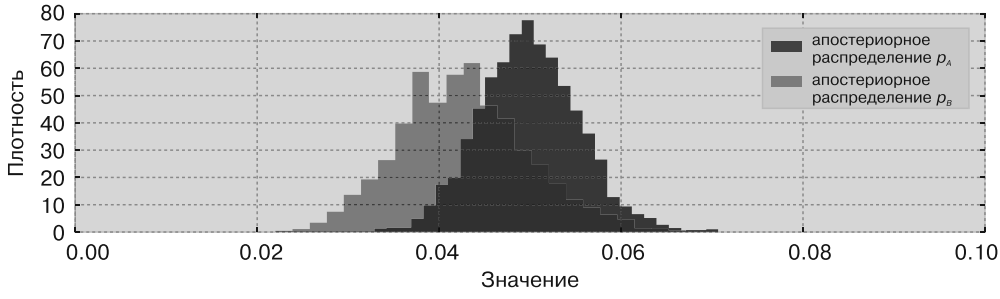

Рис. 2.7. Апостериорные распределения неизвестных p_A и p_B

С учетом апостериорного распределения delta на рис. 2.7 видно, что основная часть распределения находится выше уровня delta = 0, а значит, эффективность сайта A лучше, чем сайта B. Можно легко вычислить вероятность того, что этот вывод неверен:

```
# Подсчитываем число выборок < 0, то есть площадь под кривой до 0,
# отражающую вероятность того, что сайт A хуже сайта B.
print "Вероятность того, что сайт A ХУЖЕ сайта B: %.3f" % \
    (delta_samples < 0).mean()
print "Вероятность того, что сайт A ЛУЧШЕ сайта B: %.3f" % \
    (delta_samples > 0).mean()
```

```
[Output]:

Вероятность того, что сайт A ХУЖЕ сайта B: 0.102
Вероятность того, что сайт A ЛУЧШЕ сайта B: 0.897
```

Если эта вероятность слишком велика для того, чтобы спокойно принять решение, можно провести дополнительные испытания для сайта B (поскольку число выборок для него с самого начала было меньше, то каждая дополнительная точка данных для сайта B вносит больший вклад в вывод, чем каждая дополнительная точка данных для сайта A).

Попробуйте поэкспериментировать с параметрами `true_p_A`, `true_p_B`, `N_A` и `N_B` и посмотрите, как выглядят апостериорные распределения delta. Обратите внимание, что мы ни разу не упоминали о различии между размерами выборок сайтов A и B; они естественным образом вписываются в байесовский анализ.

Я надеюсь, читатели почувствовали, что A/B-тестирование является более естественным по сравнению с проверкой статистических гипотез, на практике зачастую только запутывающей, а не приносящей реальную пользу. В главе 7 мы увидим два расширенных варианта этой модели: один — для динамического выбора хороших сайтов, другой — для повышения скорости вычислений путем сведения анализа к одному уравнению.

2.2.5. Пример: алгоритм обнаружения мошенничества

В данных социологических исследований есть еще один интересный пласт. Люди не всегда правдивы в своих ответах, что еще более усложняет получение выводов. Например, если просто спросить испытуемых: «Мошенничали ли вы когда-нибудь на экзамене?» — то полученные ответы наверняка будут не вполне честны. Наверняка можно сказать лишь то, что фактическая частота будет меньше наблюдаемой (в предположении, что респонденты лгут *только* о том, что *не мошенничали*; мне сложно представить себе респондента, который бы утверждал, что мошенничал, если на самом деле вел себя честно).

Чтобы показать изящный способ обхода этой проблемы нечестности и продемонстрировать байесовское моделирование, я сначала познакомлю вас с биномиальным распределением.

2.2.6. Биномиальное распределение

Биномиальное распределение — одно из самых популярных распределений, в основном из-за его простоты и удобства. В отличие от других встречавшихся в этой книге распределений, у биномиального два параметра: N — положительное целочисленное значение, отражающее количество испытаний или количество экземпляров потенциальных событий, и p — вероятность появления события в отдельном испытании. Аналогично пуассоновскому распределению оно является дискретным, но, в отличие от пуассоновского, в нем задаются вероятности только для чисел от 0 до N (в пуассоновском — от 0 до бесконечности). Функция распределения масс выглядит следующим образом:

$$P(X = k) = \binom{N}{k} p^k (1-p)^{N-k}.$$

Если X — биномиальная случайная переменная с параметрами N и p, что обозначается как $X \sim \text{Bin}(N, p)$, то X равна числу событий, произошедших при N испытаниях ($0 \le X \le N$). Чем больше p (в пределах от 0 до 1), тем больше событий, вероятно, произойдет. Математическое ожидание биномиального распределения равно N_p. Построим распределение вероятностей при различных значениях параметров (рис. 2.8).

```
figsize(12.5, 4)

import scipy.stats as stats
binomial = stats.binom
parameters = [(10, .4), (10, .9)]
colors = ["#348ABD", "#A60628"]

for i in range(2):
```

```
    N, p = parameters[i]
    _x = np.arange(N + 1)
    plt.bar(_x - 0.5, binomial.pmf(_x, N, p), color=colors[i],
            edgecolor=colors[i],
            alpha=0.6,
            label="$N$: %d, $p$: %.1f" % (N, p),
            linewidth=3)

plt.legend(loc="upper left")
plt.xlim(0, 10.5)
plt.xlabel("$k$")
plt.ylabel("$P(X = k)$")
plt.title(u"Распределения вероятностей биномиальных случайных переменных");
```

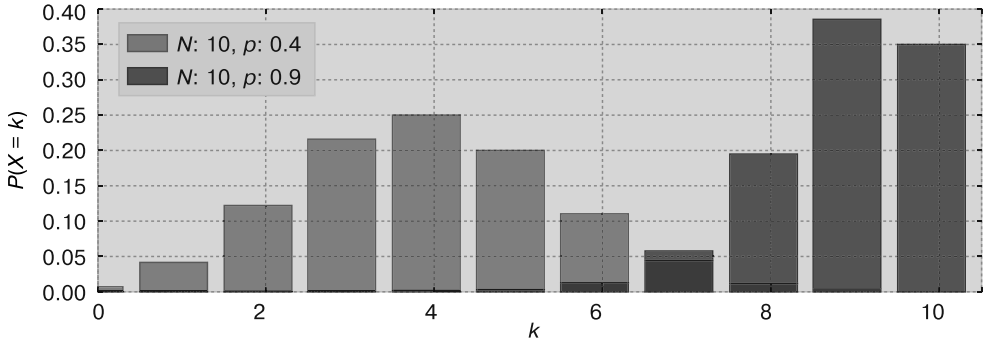

Рис. 2.8. Распределения вероятностей биномиальных случайных переменных

Частный случай с $N = 1$ соответствует распределению Бернулли. Бернуллиевы и биномиальные случайные переменные связывает и еще кое-что. Если даны бернуллиевы случайные переменные X_1, X_2... X_N с одинаковым параметром p, то $Z = X_1 + X_2 + ... + X_N \sim \text{Binomial}(N, p)$.

2.2.7. Пример: мошенничество среди студентов

Воспользуемся биномиальным распределением для определения процента студентов, мошенничавших во время экзамена. Пусть N — общее число сдававших экзамен студентов. После экзамена всех студентов опрашивают (причем результаты опроса не влекут за собой никаких последствий), и мы получаем целое число X ответов «Да, я мошенничал». Далее можно найти апостериорное распределение p при заданных N, некотором априорном распределении p и данных наблюдений X.

Это совершенно абсурдная модель. Никакой студент даже при полной безнаказанности не сознается в мошенничестве. Нам нужен более совершенный *алгоритм* опроса студентов относительно мошенничества. Желательно, чтобы он поощрял

честность в ответах при полной конфиденциальности. Ниже приведен алгоритм, который восхитил меня своей неординарностью и эффективностью [3]:

> «В процессе опроса студентов каждый студент подбрасывает (скрытно от опрашивающего) монету. Если выпадает орел, то студент отвечает честно. В противном случае (если выпадает решка) студент (скрытно) подбрасывает монету опять и отвечает: "Да, я мошенничал" — если выпадает орел, и "Нет, я не мошенничал" — если выпадает решка. При этом опрашивающий не знает, был ли ответ "Да" результатом чистосердечного признания, или орлом, выпавшим при втором подбрасывании монеты. Таким образом, конфиденциальность сохраняется, а исследователи получают правдивые ответы».

Я называю его конфиденциальным алгоритмом (для краткости: конф-алгоритмом). Можно спорить, что опрашивающие все равно получают ложные данные, поскольку часть ответов «Да» не признания, а результат случайного процесса. Иными словами, исследователи теряют примерно половину исходного набора данных, поскольку половина ответов случайны. С другой стороны, они получили хорошо моделируемый систематический процесс генерации данных. Более того, отсутствует необходимость учитывать (вероятно, немного наивно) возможность неискренних ответов. А далее можно с помощью PyMC обработать эту модель с «помехами» и найти апостериорное распределение фактической частоты лжецов.

Пусть на предмет мошенничества опрашивается 100 студентов, и нам нужно найти p — процент мошенничавших. Смоделировать это в PyMC можно несколькими способами. Я покажу сначала наиболее очевидный способ, а позднее — упрощенную версию. Выводы в обеих версиях получаются одинаковые. В нашей модели генерации данных мы производим выборку p — фактической доли мошенников — из априорного распределения. Поскольку никакой информации относительно переменной p у нас нет, мы выберем для нее априорное распределение Uniform(0, 1).

```
import pymc as pm

N = 100
p = pm.Uniform("freq_cheating", 0, 1)
```

Опять же, с учетом нашей модели генерации данных используем для 100 студентов бернуллиевы случайные переменные: 1 означает, что студент смошенничал, а 0 — что нет.

```
true_answers = pm.Bernoulli("truths", p, size=N)
```

Если продолжить выполнение этого алгоритма, то следующим шагом будет первое подбрасывание монеты студентами. Опять же, его можно промоделировать путем выборки 100 бернуллиевых случайных переменных с $p = 1 / 2$, где 1 обозначает орел, а 0 — решку.

```
first_coin_flips = pm.Bernoulli("first_flips", 0.5, size=N)
print first_coin_flips.value
```

```
[Output]:

[False False True  True  True  False True  False True  True  True  True
 False False False True  True  True  True  False True  False True  False
 True  True  False False True  True  False True  True  True  False False
 False False False True  False True  True  False False False True  True
 True  False False True  False True  True  False False False False True
 False True  True  False False False False True  False True  False False
 True  True  False True  True  False True  True  False True  False False
 True  True  False True  True  False True  True  False True  False True
 True  True  True  True]
```

Хотя *не все* студенты подбрасывают монету второй раз, смоделировать вероятную реализацию второго подбрасывания монеты все равно можно:

```
second_coin_flips = pm.Bernoulli("second_flips", 0.5, size=N)
```

С помощью этих переменных можно вернуть вероятную реализацию *наблюдаемой доли* ответов «Да». Для этого мы воспользуемся детерминистической переменной PyMC:

```
@pm.deterministic
def observed_proportion(t_a=true_answers,
                        fc=first_coin_flips,
                        sc=second_coin_flips):
    observed = fc*t_a + (1-fc)*sc
    return observed.sum() / float(N)
```

Строка `fc*t_a + (1-fc)*sc` содержит самую суть конф-алгоритма. Элементы в этом массиве равны 1 *тогда и только тогда, когда*: 1) в результате первого подбрасывания монеты выпал орел и студент жульничал на экзамене; 2) в результате первого подбрасывания монеты выпала решка, а второго — орел; в противном случае они равны 0. Наконец, в последней строке вычисляется сумма вектора и делится на `float(N)`, в результате чего возвращается процентное соотношение.

```
observed_proportion.value
```

```
[Output]:

0.26000000000000001
```

Далее нам нужен набор данных. После проведения опросов с подбрасываниями монет исследователи получили 35 ответов «Да». Относительно нашей ситуации выходит: если бы на самом деле жульничавших не было, можно было бы ожидать в среднем, что 1/4 всех ответов будут «Да» (равная 1/2 вероятность выпадения решки в первый раз и равная 1/2 вероятность выпадения орла во второй), так что

таких ответов при отсутствии жульничавших было бы около 25. С другой стороны, если бы *все студенты жульничали*, можно было бы ожидать, что около 3/4 всех ответов будут положительными.

Мы наблюдаем биномиальную случайную переменную с $N = 100$ и $p =$ `observed_proportion`, а `value` = 35:

```
X = 35
observations = pm.Binomial("obs", N, observed_proportion, observed=True,
                           value=X)
```

Далее мы добавляем все интересующие нас переменные в контейнер `Model` и запускаем для нее наш алгоритм а-ля «черный ящик».

```
model = pm.Model([p, true_answers, first_coin_flips,
                  second_coin_flips, observed_proportion, observations])

# Будет объяснено в главе 3
mcmc = pm.MCMC(model)
mcmc.sample(40000, 15000)
```

```
[Output]:

[-----------------100%-----------------] 40000 of 40000 complete in 18.7 sec
```

```
figsize(12.5, 3)
p_trace = mcmc.trace("freq_cheating")[:]
plt.hist(p_trace, histtype="stepfilled", normed=True,
         alpha=0.85, bins=30, label=u"апостериорное распределение",
         color="#348ABD")
plt.vlines([.05, .35], [0, 0], [5, 5], alpha=0.3)
plt.xlim(0, 1)
plt.xlabel(u"Значение $p$")
plt.ylabel(u"Плотность")
plt.title(u"Апостериорное распределение параметра $p$")
plt.legend();
```

Даже с учетом рис. 2.9 мы все равно не вполне уверены, какова фактическая частота жульничавших студентов, хотя и сузили ее до возможного диапазона между 0,05 и 0,35 (отмечен на рисунке сплошными вертикальными линиями). Это довольно неплохой результат, ведь *априори* мы понятия не имели, сколько студентов могло жульничать (поэтому и использовали равномерное априорное распределение). С другой стороны, не так уж этот результат и хорош: фактическое значение находится где-то в пределах окна шириной 0,3. Добились ли мы хоть чего-то или все еще не уверены в фактической частоте?

Могу поспорить, что да, мы действительно получили какую-то новую информацию. В соответствии с нашим апостериорным распределением невозможно, чтобы все студенты сдали экзамен честно, то есть согласно апостериорному распределению вероятность $p = 0$ очень низка. Мы начали с равномерного априорного распределе-

ния и считали все значения p равновероятными, однако сами данные исключили возможность $p = 0$, так что мы уверены: мошенники на экзамене были.

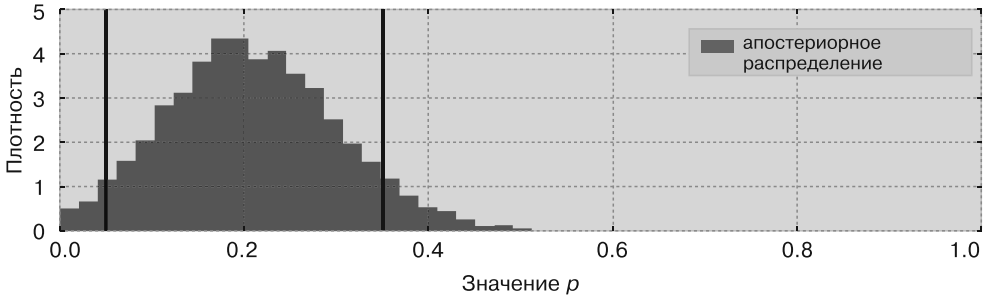

Рис. 2.9. Апостериорное распределение параметра p

Подобные алгоритмы подходят для сбора конфиденциальной информации и обеспечивают *разумную* степень уверенности в правдивости, хотя и «зашумленности» данных.

2.2.8. Альтернативная модель PyMC

При заданном значении p (которое мы, как всеведущие, знаем) можно найти вероятность получения от студента положительного ответа:

$$P(\text{«Да»}) = P(\text{выпадение орла при первом подбрасывании монеты})P(\text{мошенник}) + P(\text{выпадение решки при первом подбрасывании})P(\text{выпадение орла при втором подбрасывании}) = \frac{1}{2}p + \frac{1}{2}\frac{1}{2} = \frac{p}{2} + \frac{1}{4}.$$

Следовательно, если нам известно p, то известна и вероятность получения от студента положительного ответа. Создадим в PyMC детерминистическую функцию для вычисления вероятности ответа «Да» по заданному значению p:

```
p = pm.Uniform("freq_cheating", 0, 1)

@pm.deterministic
def p_skewed(p=p):
    return 0.5*p + 0.25
```

Можно было написать `p_skewed = 0.5*p + 0.25`, вместо того чтобы создавать однострочную функцию, ведь элементарные операции сложения и скалярного умножения приводят к неявному созданию детерминистической переменной, но я хотел ради большей понятности написать подробный стереотипный код.

Если же нам известна вероятность того, что опрашиваемый студент ответит «Да» (`p_skewed`) и мы опрашиваем $N = 100$ студентов, то число ответов «Да» представляет собой биномиальную случайную переменную с параметрами `N` и `p_skewed`.

Именно здесь мы и воспользуемся нашими зафиксированными 35 ответами «Да». При вызове `pm.Binomial` мы указываем `value=35` и `observed=True`.

```
yes_responses = pm.Binomial("number_cheaters", 100, p_skewed,
                            value=35, observed=True)
```

Далее добавляем все интересующие нас переменные в контейнер `Model` и запускаем для модели наш алгоритм а-ля «черный ящик». Получившееся апостериорное распределение показано на рис. 2.10.

```
model = pm.Model([yes_responses, p_skewed, p])

# Будет объяснено в главе 3
mcmc = pm.MCMC(model)
mcmc.sample(25000, 2500)
```

```
[Output]:

[-----------------100%-----------------] 25000 of 25000 complete in 2.0 sec
```

```
figsize(12.5, 3)
p_trace = mcmc.trace("freq_cheating")[:]
plt.hist(p_trace, histtype="stepfilled", normed=True,
         alpha=0.85, bins=30, label=u"апостериорное распределение",
         color="#348ABD")
plt.vlines([.05, .35], [0, 0], [5, 5], alpha=0.2)
plt.xlim(0, 1)
plt.xlabel(u"Значение $p$")
plt.ylabel(u"Плотность")
plt.title(u"Апостериорное распределение параметра $p$
          из альтернативной модели")
plt.legend();
```

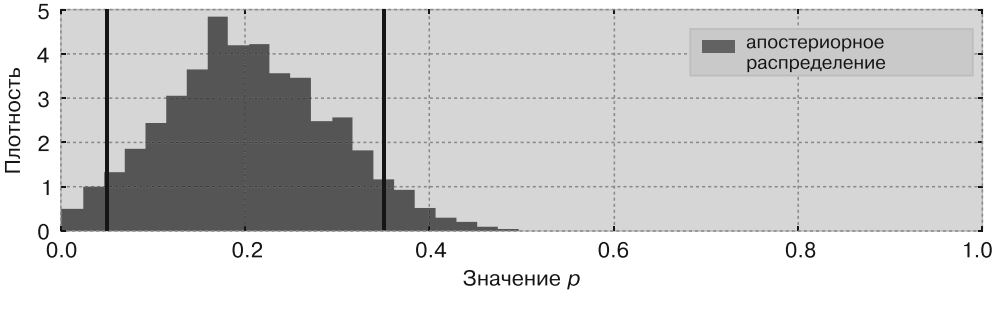

Рис. 2.10. Апостериорное распределение параметра *p* из альтернативной модели

2.2.9. Еще несколько хитростей PyMC

Совет от профи: упрощение создания детерминистических переменных с помощью класса Lambda

Иногда написание детерминистических функций с помощью декоратора `@pm.deterministic` кажется неприятной рутинной работой, особенно для небольших функций. Я уже упоминал, что элементарные математические операции *могут* неявно создавать детерминистические переменные. А что насчет таких операций, как обращение по индексу и срезы? Встроенные функции `Lambda` способны реализовать это просто и изящно. Например:

```
beta = pm.Normal("coefficients", 0, size=(N, 1))
x = np.random.randn((N, 1))
linear_combination = pm.Lambda(lambda x=x, beta=beta: np.dot(x.T, beta))
```

Совет от профи: массивы переменных PyMC

Не существует причин, мешающих сохранению нескольких однородных переменных PyMC в массиве NumPy. Не забудьте только задать при инициализации массива следующее: `dtype = object`. Например:

```
N = 10
x = np.empty(N, dtype=object)
for i in range(0, N):
     x[i] = pm.Exponential('x_%i' % i, (i+1)**2)
```

В оставшейся части этой главы мы рассмотрим несколько реальных примеров моделирования с помощью PyMC.

2.2.10. Пример: катастрофа космического челнока «Челленджер»

Двадцать восьмого января 1986 года 25-й полет в рамках космической программы США «Спейс шаттл» завершился катастрофой, когда один из ускорителей космического челнока «Челленджер» взорвался вскоре после старта, в результате чего погибли все семь членов экипажа. Президентская комиссия по расследованию пришла к выводу, что причиной катастрофы стал отказ уплотнительного кольца возле одного из стыков ускорителя в результате несовершенства конструкции. Уплотнительное кольцо стало неприемлемо чувствительным к различным факторам, в том числе к наружной температуре. Из предыдущих 24 полетов в 23 были данные об отказах уплотнительных колец, и эти данные обсуждались вечером накануне запуска «Челленджера», но, к сожалению, обсуждавшие сочли важными только данные по семи полетам, в результате которых шаттлы получили повреждения, так что было решено, что явной закономерности нет.

Эти данные используются в следующем примере кода; данные и задача изначально были сформулированы Маклишем и Струтерсом (2012) [4] и представляют собой переформулировку задачи из [5]. На рис. 2.11 показан график зависимости возникновения нештатных ситуаций относительно наружной температуры, чтобы вы могли составить себе приблизительное представление об их взаимосвязи. (Данные можно найти в репозитории GitHub: https://github.com/CamDavidsonPilon/Probabilistic-Programming-and-Bayesian-Methods-for-Hackers/blob/master/Chapter2_MorePyMC/data/challenger_data.csv.)

```
figsize(12.5, 3.5)
np.set_printoptions(precision=3, suppress=True)
challenger_data = np.genfromtxt("data/challenger_data.csv",
                                skip_header=1, usecols=[1, 2],
                                missing_values="NA",
                                delimiter=",")
# Отбрасываем значения NA.
challenger_data = challenger_data[~np.isnan(challenger_data[:, 1])]

# Строим график как функцию температуры (первый столбец).
print "Температура (F), отказ уплотнительного кольца?"
print challenger_data
plt.scatter(challenger_data[:, 0], challenger_data[:, 1], s=75,
            color="k", alpha=0.5)
plt.yticks([0, 1])
plt.ylabel("Случай повреждения?")
plt.xlabel("Наружная температура (по Фаренгейту)")
plt.title("Отказы уплотнительных колец как функция от температуры");
```

```
[Output]:

Температура (F), отказ уплотнительного кольца?
[[ 66.  0.]
 [ 70.  1.]
 [ 69.  0.]
 [ 68.  0.]
 [ 67.  0.]
 [ 72.  0.]
 [ 73.  0.]
 [ 70.  0.]
 [ 57.  1.]
 [ 63.  1.]
 [ 70.  1.]
 [ 78.  0.]
 [ 67.  0.]
 [ 53.  1.]
 [ 67.  0.]
 [ 75.  0.]
 [ 70.  0.]
 [ 81.  0.]
 [ 76.  0.]
```

```
 [ 79. 0.]
 [ 75. 1.]
 [ 76. 0.]
 [ 58. 1.]]
```

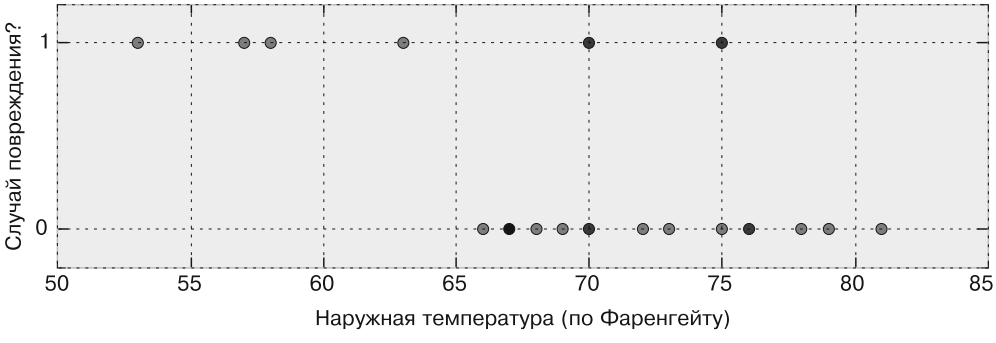

Рис. 2.11. Отказы уплотнительных колец как функция от температуры

Совершенно очевидно, что *вероятность* случаев повреждения повышается при росте наружной температуры. Здесь интересно смоделировать вероятность, потому что не похоже, чтобы была строгая зависимость между температурой и происходящими случаями повреждения. Самое большее, что можно сделать, — задать себе вопрос: «Какова вероятность повреждения при температуре *t*?» В ответе на этот вопрос и состоит цель данного примера.

Нам понадобится функция температуры — назовем ее $p(t)$, — значения которой располагаются в диапазоне от 0 до 1 (для моделирования вероятности) и меняются от 1 до 0 при повышении температуры. Подобных функций очень много, но чаще всего используется *логистическая функция* (logistic function).

$$p(t) = \frac{1}{1 + e^{\beta t}}.$$

В этой модели β — переменная, в значении которой мы не уверены. На рис. 2.12 построен график функции для $\beta = 1, 3, -5$.

```
figsize(12, 3)

def logistic(x, beta):
    return 1.0 / (1.0 + np.exp(beta * x))

x = np.linspace(-4, 4, 100)
plt.plot(x, logistic(x, 1), label=r"$\beta = 1$")
plt.plot(x, logistic(x, 3), label=r"$\beta = 3$")
plt.plot(x, logistic(x, -5), label=r"$\beta = -5$")
plt.xlabel("$x$")
```

```
plt.ylabel(u"Логистическая функция от $x$")
plt.title(u"Логистическая функция при различных значениях $\\beta$")
plt.legend();
```

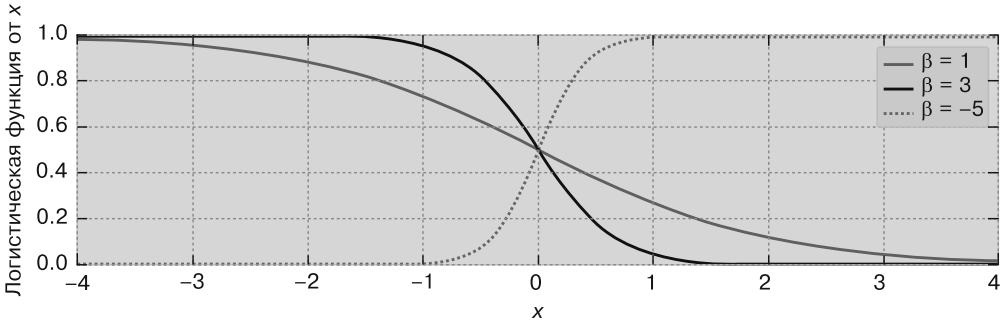

Рис. 2.12. Логистическая функция при различных значениях β

Но кое-что мы упустили. На графике логистической функции (см. рис. 2.12) вероятность меняется только около 0, но в наших данных по «Челленджеру», показанных на рис. 2.11, вероятность меняется при значениях 65–70 градусов Фаренгейта. Необходимо ввести в уравнение логистической функции *смещение* (bias):

$$p(t) = \frac{1}{1 + e^{\beta t + \alpha}}.$$

На рис. 2.13 показана наша логистическая функция при различных значениях α.

```
def logistic(x, beta, alpha=0):
    return 1.0 / (1.0 + np.exp(np.dot(beta, x) + alpha))

x = np.linspace(-4, 4, 100)

plt.plot(x, logistic(x, 1), label=r"$\beta = 1$", ls="--", lw=1)
plt.plot(x, logistic(x, 3), label=r"$\beta = 3$", ls="--", lw=1)
plt.plot(x, logistic(x, -5), label=r"$\beta = -5$", ls="--", lw=1)

plt.plot(x, logistic(x, 1, 1), label=r"$\beta = 1, \alpha = 1$",
         color="#348ABD")
plt.plot(x, logistic(x, 3, -2), label=r"$\beta = 3, \alpha = -2$",
         color="#A60628")
plt.plot(x, logistic(x, -5, 7), label=r"$\beta = -5, \alpha = 7$",
         color="#7A68A6")

plt.title(u"Логистическая функция при различных значениях $\\beta$ и\
          $\\alpha$")
plt.xlabel("$x$")
plt.ylabel(u"Логистическая функция от $x$")
plt.legend(loc="lower left");
```

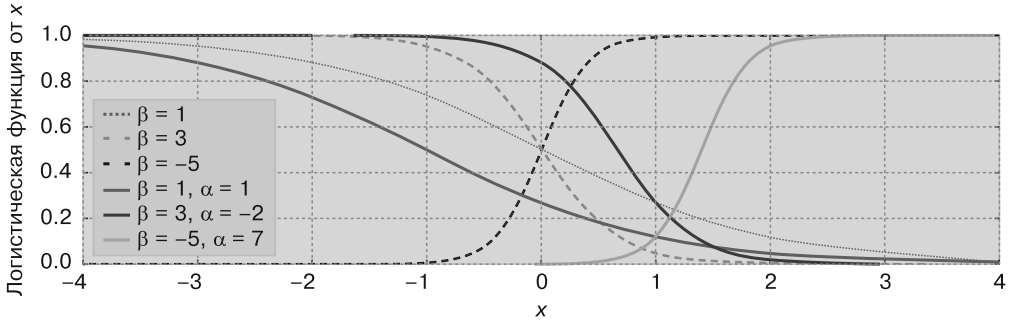

Рис. 2.13. Логистическая функция при различных значениях β и α

Добавление свободного члена α соответствует сдвигу кривой влево или вправо (отсюда и название *«смещение»*).

Приступим к моделированию этого в PyMC. Параметры β и α не обязаны быть положительными, чем-то ограниченными или относительно большими, так что лучше всего моделировать их с помощью *нормальных случайных переменных* (normal random variable), с которыми я познакомлю вас в следующем подразделе.

2.2.11. Нормальное распределение

У нормальных случайных переменных, обозначаемых $X \sim N(\mu, 1/\tau)$, распределение параметризуется средним значением μ и *точностью* (precision) τ. Те, кто уже знаком с нормальным распределением, возможно, встречали σ^2 вместо τ^{-1}. На самом деле это обратные друг другу величины. Я внес такое изменение, чтобы упростить математический анализ. Просто запомните: чем меньше τ, тем шире распределение (то есть степень уверенности ниже); чем больше τ, тем распределение у́же (то есть степень нашей уверенности выше). В любом случае τ всегда больше нуля.

Функция плотности вероятности случайной переменной $N(\mu, 1/\tau)$ равна:

$$f(x \mid \mu, \tau) = \sqrt{\frac{\tau}{2\pi}} \exp(-\frac{\tau}{2}(x-\mu)^2).$$

Построим несколько графиков различных функций плотности нормального распределения (рис. 2.14).

```
import scipy.stats as stats

nor = stats.norm
x = np.linspace(-8, 7, 150)
mu = (-2, 0, 3)
tau = (.7, 1, 2.8)
colors = ["#348ABD", "#A60628", "#7A68A6"]
```

```
parameters = zip(mu, tau, colors)

for _mu, _tau, _color in parameters:
    plt.plot(x, nor.pdf(x, _mu, scale=1./_tau),
             label="$\mu = %d,\;\\tau = %.1f$" % (_mu, _tau),
             color=_color)
    plt.fill_between(x, nor.pdf(x, _mu, scale=1./_tau), color=_color, alpha=.33)

plt.legend(loc="upper right")
plt.xlabel("$x$")
plt.ylabel(u"Функция плотности от $x$")
plt.title(u"Распределение вероятности трех различных нормальных случайных \
          переменных");
```

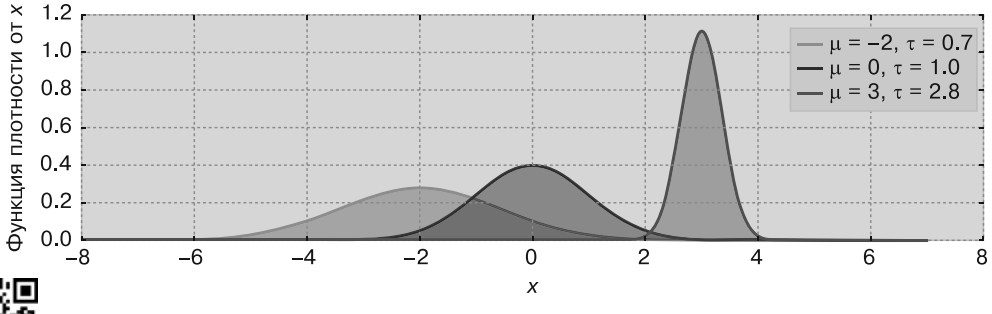

Рис. 2.14. Распределение вероятности трех различных нормальных случайных переменных

Нормальная случайная переменная может принимать любое вещественное значение, хотя очень вероятно, что оно будет близко к μ. На самом деле математическое ожидание нормального распределения равно его параметру μ:

$$E[X|\mu, \tau] = \mu,$$

а его дисперсия обратно пропорциональна μ:

$$Var(X|\mu, \tau) = \frac{1}{\tau}.$$

Продолжим моделирование полета космического корабля «Челленджер».

```
Import pymc as pm

temperature = challenger_data[:, 0]
D = challenger_data[:, 1] # повреждение или нет?

# Обратите внимание на "value". Я поясню это позднее.
Beta = pm.Normal("beta", 0, 0.001, value=0)
```

```
alpha = pm.Normal("alpha", 0, 0.001, value=0)

@pm.deterministic
def p(t=temperature, alpha=alpha, beta=beta):
    return 1.0 / (1. + np.exp(beta*t + alpha))
```

Теперь у нас есть нужные распределения вероятностей, но как связать их с данными наблюдений? Воспользуемся *бернуллиевой* случайной переменной, которую мы рассматривали в подразделе 2.2.3. Таким образом, наша модель будет выглядеть так:

$$\text{Случай повреждения, } D_i \sim \text{Ber}(p(t_i)),\ i = 1, \ldots N,$$

где $p(t)$ — логистическая функция (чьи значения располагаются строго между 0 и 1), а t_i — температуры, для которых у нас есть наблюдения. Отмечу, что в данном коде нам пришлось присвоить `beta` и `alpha` значение `0`. Дело в том, что очень большие значения `beta` и `alpha` приводят к тому, что p будет равно 1 или 0. К сожалению, метод `pm.Bernoulli` «не любит» вероятностей, в точности равных 0 или 1, хотя с точки зрения математики они вполне допустимы. Так что благодаря установке значений коэффициентов равными 0 мы получаем приемлемое начальное значение переменной p. Это никак не влияет на результаты и не означает учета дополнительной информации в априорном распределении. Это просто особенность вычислений в PyMC.

```
p.value
```

```
[Output]:

array([ 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
        0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5, 0.5,
        0.5])
```

```
# Связываем вероятности "p" с данными наблюдений с помощью
# бернуллиевой случайной переменной.
Observed = pm.Bernoulli("bernoulli_obs", p, value=D, observed=True)
model = pm.Model([observed, beta, alpha])

# Загадочный код, который я поясню в главе 3
map_ = pm.MAP(model)
map_.fit()
mcmc = pm.MCMC(model)
mcmc.sample(120000, 100000, 2)
```

```
[Output]:

[-----------------100%-----------------] 120000 of 120000 complete
    in 15.3 sec
```

Мы обучили модель на данных наблюдений. Теперь можно произвести выборку значений из апостериорного распределения. Взглянем на апостериорные распределения для α и β, показанные на рис. 2.15.

```
alpha_samples = mcmc.trace('alpha')[:, None]
# лучше, чтобы они были одномерными
beta_samples = mcmc.trace('beta')[:, None]

figsize(12.5, 6)

# Гистограмма выборок
plt.subplot(211)
plt.title(ur"Апостериорные распределения параметров модели \
          $\alpha, \beta$")
plt.hist(beta_samples, histtype='stepfilled', bins=35, alpha=0.85,
         label=ur"апостериорное распределение $\beta$",
         color="#7A68A6", normed=True)
plt.legend()

plt.subplot(212)
plt.hist(alpha_samples, histtype='stepfilled', bins=35, alpha=0.85,
        label=ur"апостериорное распределение $\alpha$", color="#A60628",
        normed=True)
plt.xlabel(u"Значение параметра")
plt.xlabel(u"Плотность")
plt.legend();
```

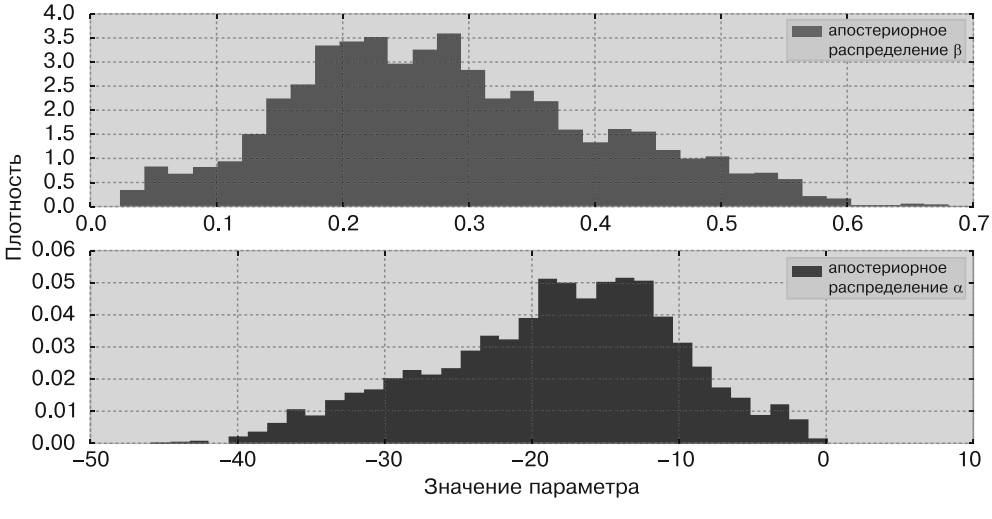

Рис. 2.15. Апостериорные распределения параметров модели α и β

Все выборки β превышают 0. Если бы апостериорное распределение, напротив, было сосредоточено вокруг 0, то можно было бы заподозрить, что $\beta = 0$, а значит, температура никак не влияет на вероятность повреждения. Аналогично все апо-

стериорные значения α отрицательны и находятся далеко от 0, так что можно обоснованно предположить, что α значительно меньше 0. Несмотря на разброс данных, уверенности в фактических значениях параметров у нас практически нет (хотя такое поведение вполне ожидаемо, если учесть малый размер выборки и значительное перекрытие повреждений и не-повреждений).

Теперь вычислим *ожидаемую вероятность* для конкретного значения температуры. То есть выполним усреднение по всем выборкам из апостериорного распределения, чтобы получить вероятное значение $p(t_i)$ (рис. 2.16).

```
t = np.linspace(temperature.min() - 5, temperature.max()+5, 50)[:, None]
p_t = logistic(t.T, beta_samples, alpha_samples)

mean_prob_t = p_t.mean(axis=0)
figsize(12.5, 4)

plt.plot(t, mean_prob_t, lw=3, label=u"средняя апостериорная \nвероятность\
         повреждения")
plt.plot(t, p_t[0, :], ls="--", label=u"реализация из апостериорного\
         распределения")
plt.plot(t, p_t[-2, :], ls="--", label=u"реализация из апостериорного\
         распределения")
plt.scatter(temperature, D, color="k", s=50, alpha=0.5)
plt.title(u"Апостериорное математическое ожидание вероятности повреждения,\
          в том числе две реализации")
plt.legend(loc="lower left")
plt.ylim(-0.1, 1.1)
plt.xlim(t.min(), t.max())
plt.ylabel(u"Вероятность")
plt.xlabel(u"Температура");
```

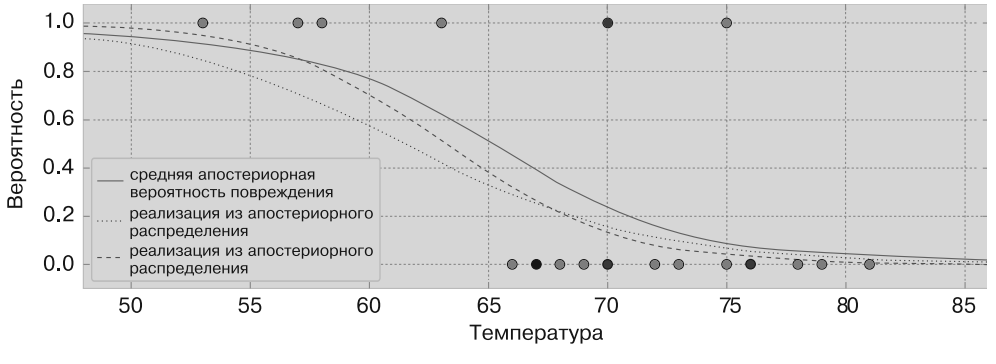

Рис. 2.16. Апостериорное математическое ожидание вероятности повреждения, в том числе две реализации

На рис. 2.16 мы строим графики двух реализаций возможной фактической ситуации. Обе они столь же вероятны, как и любой другой вариант. Сплошная линия получается при усреднении всех 20 000 возможных пунктирных линий.

Любопытно было бы узнать, при каких температурах наша уверенность в вероятности повреждения ниже всего. На рис. 2.17 показан график кривой математического ожидания и соответствующие байесовские доверительные интервалы (credible interval, CI) с уровнем доверия 95 % для всех температур.

```
from scipy.stats.mstats import mquantiles

# Векторизованные 2,5%-ные квантили для байесовского
# доверительного интервала
qs = mquantiles(p_t, [0.025, 0.975], axis=0)
plt.fill_between(t[:, 0], *qs, alpha=0.7,
                 color="#7A68A6")
plt.plot(t[:, 0], qs[0], label="95% CI", color="#7A68A6", alpha=0.7)

plt.plot(t, mean_prob_t, lw=1, ls="--", color="k",
         label= u"средняя апостериорная \nвероятность\ повреждения")

plt.xlim(t.min(), t.max())
plt.ylim(-0.02, 1.02)
plt.legend(loc="lower left")
plt.scatter(temperature, D, color="k", s=50, alpha=0.5)
plt.xlabel(u"Температура, $t$")

plt.ylabel(u"Оценка вероятности")
plt.title(u"Апостериорная вероятность оценок при заданной температуре $t$");
```

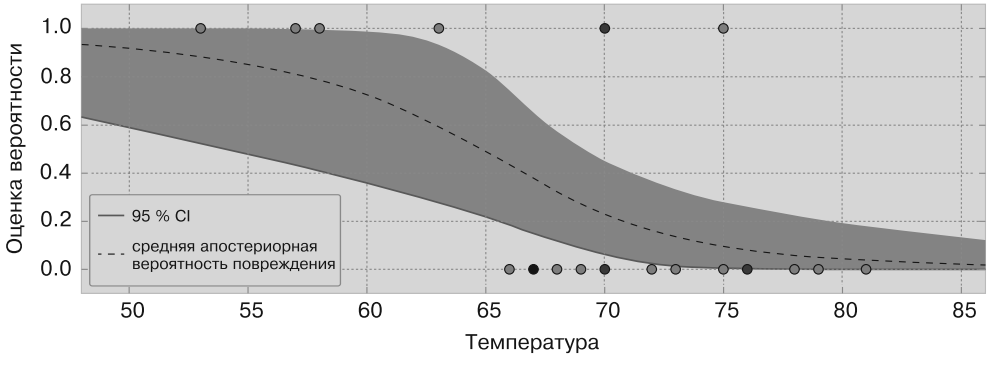

Рис. 2.17. Апостериорная вероятность оценок при заданной температуре *t*

95%-ный байесовский доверительный интервал (95 % CI), нарисованный темно-серым цветом, соответствует для каждого значения температуры интервалу, содержащему 95 % распределения. Например, при 65 градусах мы на 95 % уверены, что вероятность повреждения находится между 0,25 и 0,75. Не следует путать это понятие с *доверительным интервалом* (confidence interval) из частотного подхода, у которого несколько иной смысл.

Как видим, при температуре около 60 градусов байесовский доверительный интервал быстро расширяется в пределах отрезка [0,1]. А при пересечении от-

метки 70 градусов байесовский доверительный интервал снова сужается. Из этой информации понятно, что делать дальше: вероятно, имеет смысл протестировать больше уплотнительных колец при температурах от 60 до 65 градусов, чтобы лучше оценить вероятности в этом диапазоне. Вам также следует с осторожностью сообщать ученым свои оценки, ведь одно только значение ожидаемой вероятности не отражает *ширины* апостериорного распределения.

2.2.12. Что произошло в день катастрофы «Челленджера»

В день катастрофы «Челленджера» наружная температура была 31 градус по Фаренгейту. Каково апостериорное распределение возникновения повреждения при такой температуре? Распределение изображено на рис. 2.18. Похоже, повреждение уплотнительных колец в «Челленджере» было практически неизбежно.

```
figsize(12.5, 2.5)

prob_31 = logistic(31, beta_samples, alpha_samples)

plt.xlim(0.995, 1)
plt.hist(prob_31, bins=1000, normed=True, histtype='stepfilled')
plt.title(u"Апостериорное распределение вероятности возникновения повреждения\
          при $t = 31$")
plt.ylabel(u"Плотность")
plt.xlabel(u"Вероятность возникновения повреждения в уплотнительном кольце");
```

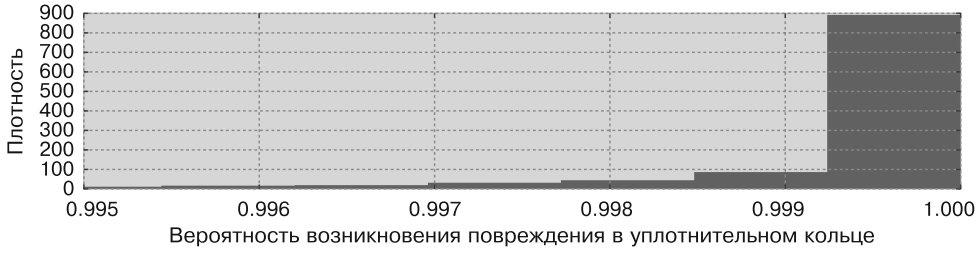

Рис. 2.18. Апостериорное распределение вероятности возникновения повреждения при $t = 31$

2.3. Адекватна ли наша модель?

Скептически настроенный читатель может заметить: «Вы специально выбрали логистическую функцию в качестве $p(t)$ и конкретные априорные распределения. Возможно, результаты при других функциях и априорных распределениях оказались бы другими. Откуда я знаю, что была выбрана хорошая модель?» Это совершенно справедливое замечание. Рассмотрим предельный случай. Что, если выбрать $p(t) = 1$ для всех t, гарантирующую возникновение повреждений при любом t? Я бы предсказал катастрофу 28 января, но тем не менее это очень плохая модель.

С другой стороны, если выбрать логистическую функцию в качестве $p(t)$, но задать сильно сконцентрированные около 0 априорные распределения, то апостериорные распределения окажутся совсем другими. Откуда мы знаем, что наша модель отражает полученные данные? Имеет смысл вычислить *критерий согласия* (goodness of fit) модели, то есть оценить, насколько хорошо она удовлетворяет наблюдениям.

Как можно проверить, не плохо ли модель удовлетворяет данным? Один из способов — сравнить наблюдаемые данные (которые, как вы помните, представляют собой *фиксированную* стохастическую переменную) с искусственным набором данных, полученным путем имитационного моделирования. Логическое обоснование такого действия состоит в том, что если имитационный набор данных окажется статистически непохожим на наблюдаемый набор, значит наша модель плохо отражает наблюдаемые данные.

Ранее в этой главе путем имитационного моделирования мы создали искусственный набор данных для примера с обменом текстовыми сообщениями. Для этого мы выбрали значения из априорных распределений (то есть выбрали значения из модели, которая не была обучена на данных). Мы видели, насколько различными оказались получившиеся наборы данных и насколько редко они походили на наблюдаемый набор данных. В текущем примере для создания правдоподобных наборов данных следует выбирать данные из апостериорных распределений. Мы просто создадим новую стохастическую переменную, точно такую же, как и переменная, в которой хранятся наши наблюдения, за исключением того, что в ней не будет самих данных наблюдений. Как вы помните, стохастическая переменная, в которой хранятся наши данные наблюдений, была такой:

```
observed = pm.Bernoulli("bernoulli_obs", p, value=D, observed=True)
```

Поэтому для создания правдоподобных наборов данных мы инициализируем следующую переменную (рис. 2.19):

```
simulated_data = pm.Bernoulli("simulation_data", p)

simulated = pm.Bernoulli("bernoulli_sim", p)
N = 10000

mcmc = pm.MCMC([simulated, alpha, beta, observed])
mcmc.sample(N)
```

```
[Output]:

[-----------------100%-----------------] 10000 of 10000 complete in 2.4 sec
```

```
figsize(12.5, 5)

simulations = mcmc.trace("bernoulli_sim")[:].astype(int)
print "Форма массива имитаций: ", simulations.shape

plt.title(u"Имитационные наборы данных на основе апостериорных параметров")
figsize(12.5, 6)
```

```
for i in range(4):
    ax = plt.subplot(4, 1, i+1)
    plt.scatter(temperature, simulations[1000*i, :], color="k", s=50, alpha=0.6);
```

```
[Output]:

Форма массива имитаций: (10000, 23)
```

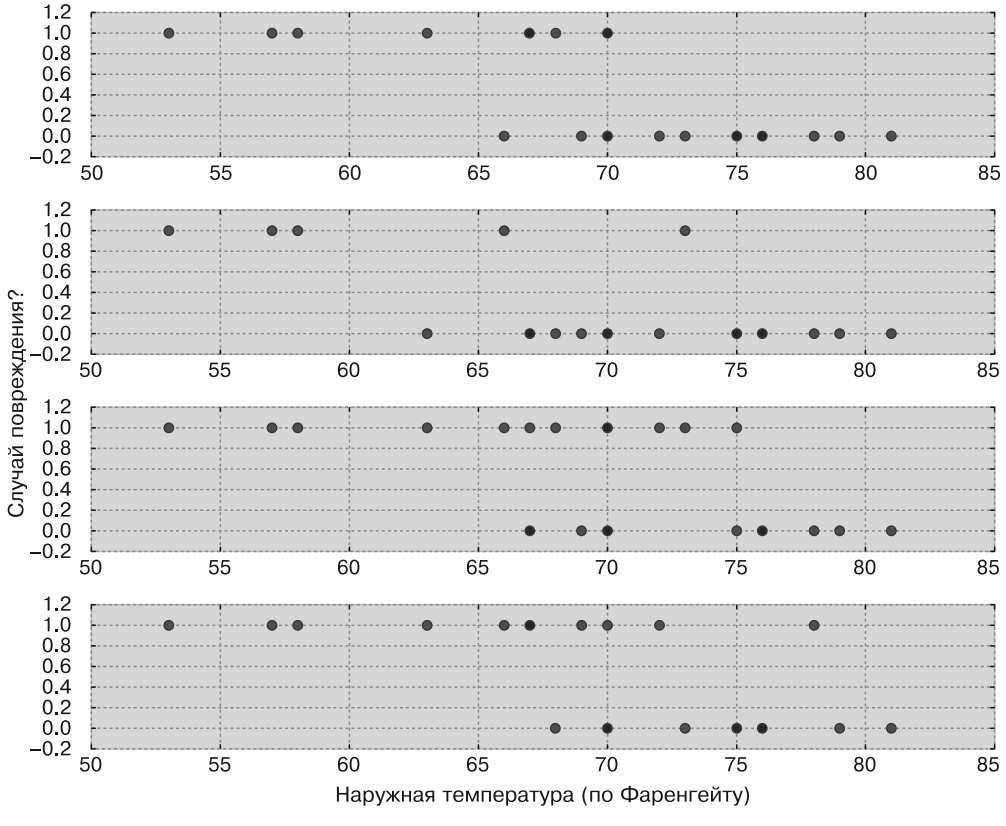

Рис. 2.19. Имитационные наборы данных на основе апостериорных параметров

Отмечу, что графики на рис. 2.19 различаются, поскольку различны лежащие в их основе данные. Однако все они получены на базе одной и той же модели: вследствие своей случайности они выглядят по-разному, но их объединяют общие статистические показатели. Похожи ли эти наборы данных (статистически) на наши исходные данные наблюдений?

Мы хотим оценить, насколько хороша наша модель. «Хороший» — понятие субъективное, конечно, так что результаты необходимо рассматривать относительно других моделей.

Сравнивать мы также будем визуально, что может показаться еще менее объективным методом. В качестве альтернативы можно было воспользоваться *байесовскими p-значениями*, которые представляют собой сводные статистические данные модели и аналогичны *p*-значениям из частотного подхода. Тем не менее байесовские *p*-значения тоже субъективны, поскольку граница между *хорошим* и *плохим* выбирается произвольно. Гелман подчеркивает, что визуальные проверки более иллюстративны [6], чем проверки с помощью *p*-значений. И мы с ним согласны.

2.3.1. Разделительные графики

Следующая визуальная проверка представляет собой новый подход к визуализации данных при обучении с помощью логистической регрессии. Такие графики называются *разделительными* (separation plots). С помощью разделительных графиков пользователь может визуально сравнить между собой наборы моделей. Я опущу большую часть технических подробностей (см. весьма понятно изложенную статью [7]), но вкратце расскажу здесь про их использование.

Для каждой из моделей вычисляется доля случаев, когда апостериорное моделирование возвращает значение 1 для конкретной температуры — то есть выполняется вычисление $P(\text{Повреждение} = 1|t)$ — с усреднением по всем имитационным моделям. В результате мы получаем апостериорную вероятность повреждения для каждой точки данных нашего набора. Например, для вышеприведенной модели:

```
posterior_probability = simulations.mean(axis=0)
print "Набл. | Массив имитационных повреждений \
             | Апостериорная вероятность повреждения | Возникший дефект"
for i in range(len(D)):
    print "%s | %s | %.2f | %d" %\
          (str(i).zfill(2),str(simulations[:10,i])[:-1] + "...]".ljust(12),
           posterior_probability[i], D[i])
```

```
[Output]:

Набл. | Массив имитационных повреждений | Апостериорная | Возникший дефект
                                          вероятность
                                          повреждения
00    | [0 0 1 0 0 1 0 0 0 1...]        | 0.45          |   0
01    | [0 1 1 0 0 0 0 0 0 1...]        | 0.22          |   1
02    | [1 0 0 0 0 0 0 0 0 0...]        | 0.27          |   0
03    | [0 0 0 0 0 0 0 1 0 1...]        | 0.33          |   0
04    | [0 0 0 0 0 0 0 0 0 0...]        | 0.39          |   0
05    | [1 0 1 0 0 1 0 0 0 0...]        | 0.14          |   0
06    | [0 0 1 0 0 0 1 0 0 0...]        | 0.12          |   0
07    | [0 0 0 0 0 0 1 0 0 1...]        | 0.22          |   0
08    | [1 1 0 0 1 1 0 0 1 0...]        | 0.88          |   1
09    | [0 0 0 0 0 0 0 0 0 1...]        | 0.65          |   1
10    | [0 0 0 0 0 1 0 0 0 0...]        | 0.22          |   1
11    | [0 0 0 0 0 0 0 0 0 0...]        | 0.04          |   0
12    | [0 0 0 0 0 1 0 0 0 0...]        | 0.39          |   0
```

```
13    | [1 1 0 0 0 1 1 0 0 1...]        | 0.95          |   1
14    | [0 0 0 0 1 0 0 1 0 0...]        | 0.39          |   0
15    | [0 0 0 0 0 0 0 0 0 0...]        | 0.08          |   0
16    | [0 0 0 0 0 0 0 0 1 0...]        | 0.23          |   0
17    | [0 0 0 0 0 0 1 0 0 0...]        | 0.02          |   0
18    | [0 0 0 0 0 0 0 1 0 0...]        | 0.06          |   0
19    | [0 0 0 0 0 0 0 0 0 0...]        | 0.03          |   0
20    | [0 0 0 0 0 0 0 1 1 0...]        | 0.07          |   1
21    | [0 1 0 0 0 0 0 0 0 0...]        | 0.06          |   0
22    | [1 0 1 1 0 1 1 1 0 0...]        | 0.86          |   1
```

Отсортируем столбцы по апостериорной вероятности.

```
ix = np.argsort(posterior_probability)
print "Апостериорная вероятность повреждения | Возникший дефект"
for i in range(len(D)):
    print "%.2f               | %d" %
          (posterior_probability[ix[i]], D[ix[i]])
```

```
[Output]:

Апостериорная вероятность повреждения | Возникший дефект
0.02                                  |   0
0.03                                  |   0
0.04                                  |   0
0.06                                  |   0
0.06                                  |   0
0.07                                  |   1
0.08                                  |   0
0.12                                  |   0
0.14                                  |   0
0.22                                  |   1
0.22                                  |   0
0.22                                  |   1
0.23                                  |   0
0.27                                  |   0
0.33                                  |   0
0.39                                  |   0
0.39                                  |   0
0.39                                  |   0
0.45                                  |   0
0.65                                  |   1
0.86                                  |   1
0.88                                  |   1
0.95                                  |   1
```

Можно отобразить эти данные более наглядно, на графике (рис. 2.20). Воспользуемся функцией `separation_plot`.

```
from separation_plot import separation_plot

figsize(11, 1.5)
separation_plot(posterior_probability, D)
```

Зигзагообразная линия представляет собой отсортированные апостериорные вероятности, темные столбики отмечают возникшие повреждения, а пустое пространство (светло-серые столбики, с точки зрения читателей-оптимистов) показывает случаи, когда повреждений не возникло. По мере роста вероятности возникает все больше повреждений. График означает, что при большой апостериорной вероятности (близости линии к 1) фактически возникает больше повреждений. Это хорошо. В идеале все темно-серые столбики *должны* быть близки к правой стороне, отклонения от такого поведения означают неудачные предсказания (прогнозы).

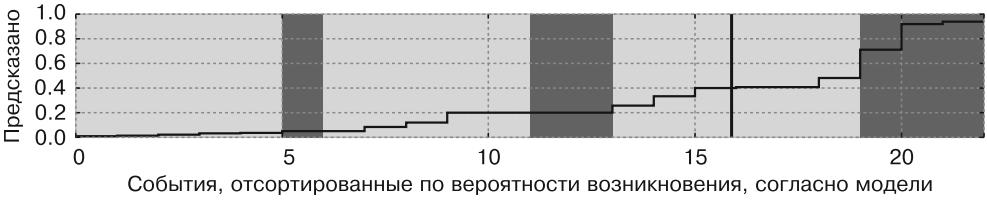

Рис. 2.20. Температурно-зависимая модель

Черная вертикальная прямая отмечает ожидаемое количество повреждений, которое мы должны наблюдать при такой модели (см. подробности вычислений в приложении 2.5). Благодаря этому пользователь может сравнить общее число предсказанных моделью событий с числом событий, фактически имевших место, судя по данным.

Гораздо полезнее будет сравнить этот разделительный график с разделительными графиками других моделей. На рис. 2.21–2.24 мы сравниваем нашу модель (сверху) с тремя другими.

1. Модель идеального предсказания, в которой апостериорная вероятность равна 1, если повреждение имело место, и 0 в противном случае.
2. Абсолютно случайная модель, предсказывающая случайные вероятности вне зависимости от температуры.
3. Константная модель, в которой $P(D = 1|t) = c, \forall t$. Лучше всего выбрать в качестве c наблюдаемую частоту повреждений — в данном случае $7 / 23$.

```
figsize(11, 1.25)

# Температурно-независимая модель
separation_plot(posterior_probability, D)
plt.title("Наша байесовская температурно-зависимая модель")

# Идеальная модель (вероятность повреждения соответствует тому,
# возникло оно в реальности или нет)
p = D
separation_plot(p, D)
```

```
plt.title(u"Идеальная модель")

# random predictions
p = np.random.rand(23)
separation_plot(p, D)
plt.title(u"Случайная модель")

# Константная модель
constant_prob = 7./23*np.ones(23)
separation_plot(constant_prob, D)
plt.title(u"Модель неизменных предсказаний");
```

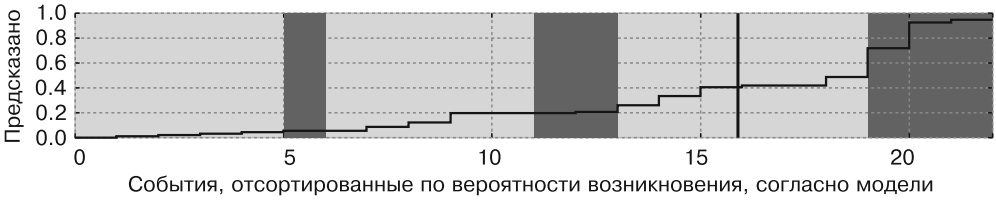

Рис. 2.21. Наша байесовская температурно-зависимая модель

Рис. 2.22. Идеальная модель

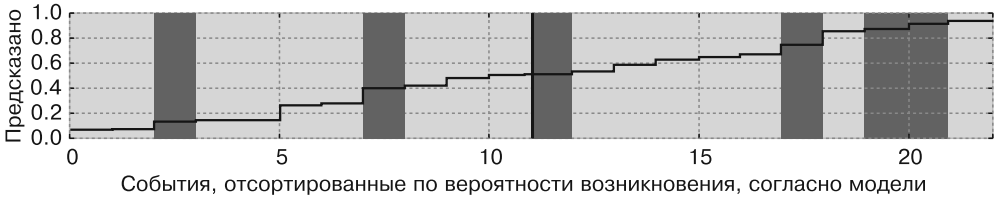

Рис. 2.23. Случайная модель

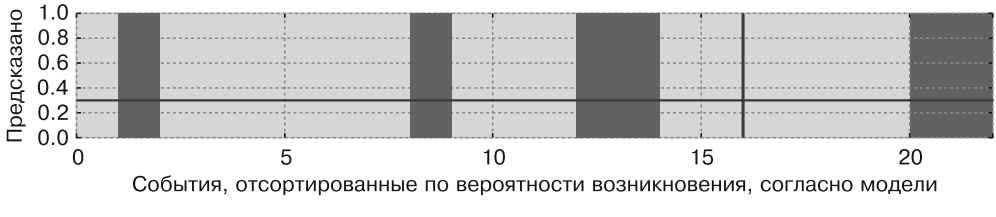

Рис. 2.24. Модель неизменных предсказаний

На графике случайной модели (см. рис. 2.23) заметно, что при росте вероятности повреждения не сосредотачиваются справа. Этим она схожа с константной моделью.

На графике идеальной модели (см. рис. 2.22) линия вероятности плохо видна, поскольку располагается в самом низу и самом верху графика. Конечно, идеальная модель может служить лишь для демонстрационных целей, никаких выводов из нее сделать нельзя.

2.4. Выводы

В этой главе мы проанализировали синтаксис создания моделей PyMC, разобрались в логике создания байесовских моделей и изучили несколько реальных примеров: A/B-тестирование, использование конф-алгоритма и катастрофу космического челнока «Челленджер».

Для моделирования в этой главе необходимы базовые знания используемых распределений, которые пригодятся и при дальнейшем байесовском моделировании. Как я уже упоминал, распределения — базовые блоки байесовского моделирования, так что стоит получше разобраться в них. Ошибки при выборе распределений — частое явление, но PyMC в этом смысле проявляет снисхождение и «громко кричит», если что-то не так. В подобном случае лучше вернуться назад и снова проанализировать свой выбор распределений.

В главе 3 мы выясним, что происходит у PyMC «под капотом», — это поможет нам при отладке в случаях ошибок при моделировании.

2.5. Приложение

Как вычислить ожидаемое число повреждений при обученной модели (в более общем случае: число ожидаемых экземпляров какой-либо категории)? Пусть у нас есть N наблюдений, каждое с какими-либо уникальными характеристиками (в нашем примере — температурой); мы можем вычислить вероятность категории (в нашем случае — *повреждения*) для каждого из наблюдений.

Каждое наблюдение с присвоенным ему индексом i можно рассматривать как бернуллиеву случайную переменную (B_i) при следующей модели: $B_i = 1$ (то есть мы правы) с вероятностью p_i и $B_i = 0$ (то есть мы ошиблись) с вероятностью $1 - p_i$, где все вероятности p_i возвращает обученная модель на основе наблюдений. Сумма всех этих бернуллиевых случайных переменных и есть итоговое число экземпляров из нужной категории при данной модели. Например, если мы специально завышаем все p_i, то сумма тоже окажется завышенной, что может отличаться от наблюдаемого на практике (где итоговое число в отдельных категориях может быть небольшим).

Ожидаемое число повреждений равно математическому ожиданию суммы:

$$S = \sum_{i=0}^{N} X_i,$$

$$E[S] = \sum_{i=0}^{N} E[X_i] = \sum_{i=0}^{N} p_i.$$

поскольку математическое ожидание бернуллиевой случайной переменной равно вероятности ее равенства 1. Таким образом, для разделительных графиков нам нужно вычислить эту сумму вероятностей и провести вертикальную линию в соответствующем месте.

Относительно перекрестной проверки: этот шаг выполняется до каких-либо оценок с помощью контрольных данных и может быть частью процесса обучения для сравнения критериев согласия различных моделей.

2.6. Упражнения

1. Попробуйте задать предельные значения для наших наблюдений в примере с мошенничеством студентов. Что будет, если ответов «Да» было 25; 10; 50?
2. Попробуйте построить графики выборок α по сравнению с выборками β. Почему полученный график выглядит таким образом?

2.6.1. Ответы

1. Допустим, утвердительных ответов не было (равно 0). Это одно из предельных значений; значит, все, кто выбросил орел при первом подбрасывании, честно вели себя на экзамене, а те, кто выбросил решку, при втором подбрасывании также выбросили решку. Если мы запустим вычисление этой модели, то обнаружим определенную апостериорную массу для значений вдалеке от 0. Почему так происходит? Причина в том, что согласно нашей схеме опыта жульничавший студент мог выбросить решку при первом броске и мы так и не услышали бы от него правдивого ответа. Наша модель решает данную проблему за счет повышения веса удаленных от 0 значений. Аналогичное поведение имеет место при 100 утвердительных ответах, но только для удаленных от 1 значений.
2.
```
figsize(12.5, 4)

plt.scatter(alpha_samples, beta_samples, alpha=0.1)
plt.title(u"Почему график выглядит подобным образом?")
plt.xlabel(r"$\alpha$")
plt.ylabel(r"$\beta$")
```

2.7. Библиография

1. *Cronin B.* Why Probabilistic Programming Matters. https://plus.google.com/u/0/107971134877020469960/posts/KpeRdJKR6Z1.
2. A Probability Distribution Value Exceeding 1 Is OK? Cross Validated. http://stats.stackexchange.com/questions/4220/a-probability-distribution-value-exceeding-1-is-ok/.
3. *Warner S. L.* Randomized Response: A Survey Technique for Eliminating Evasive Answer Bias // Journal of the American Statistical Association, 60, № 309 (Mar., 1965): 63–69.
4. *McLeish D., Struthers C.* STATISTICS 450/850: Estimation and Hypothesis Testing, Supplementary Lecture Notes. Ontario: University of Waterloo, 2013.
5. *Dalal S. R., Fowlkes E. B., Hoadley B.* Risk Analysis of the Space Shuttle: Pre-Challenger Prediction of Failure // Journal of the American Statistical Association, 84, № 408 (Dec., 1989): 945–957.
6. *Gelman A., Shalizi C. R.* Philosophy and the Practice of Bayesian Statistics // British Journal of Mathematical and Statistical Psychology, 66 (2013): 8–38.
7. *Greenhill B., Ward M. D., Sacks A.* The Separation Plot: A New Visual Method for Evaluating the Fit of Binary Models // American Journal of Political Science, 55, № 4 (2011): 991–1002.

3 Открываем «черный ящик» MCMC

3.1. Байесовский ландшафт

В главах 1 и 2 внутренние механизмы PyMC в частности и метода Монте-Карло по схеме марковской цепи (MCMC) в общем были скрыты от читателя. Для включения в книгу данной главы было три причины. Во-первых, в любой книге, посвященной байесовскому выводу, должно рассказываться про MCMC. С этим не поспоришь. Вина за это лежит на специалистах по статистике. Во-вторых, чем лучше вы ориентируетесь в процессе MCMC, тем лучше понимаете, сходится ли ваш алгоритм. (Сходится к чему? Мы еще доберемся до этого вопроса.) В-третьих, нужно выяснить, *почему* мы возвращаем в качестве решения тысячи выборок из апостериорного распределения (что, на первый взгляд, довольно странно).

При формулировке задачи байесовского вывода с N неизвестными неявным образом создается N-мерное пространство для априорных распределений. С этим пространством связано также дополнительное измерение, которое можно описать как *поверхность* (surface) или *кривую* (curve) над этим пространством, отражающую *априорную вероятность* (prior probability) конкретной точки. Поверхность пространства определяется априорными вероятностями. Например, при двух неизвестных — p_1 и p_2 — с равномерными априорными распределениями Uniform(0, 5) создаваемое пространство имеет вид квадрата со стороной 5, а поверхность представляет собой плоскость над этим квадратом (отражая тот факт, что все точки равновероятны, поскольку априорные распределения равномерные). Все это показано на рис. 3.1.

```
%matplotlib inline
import scipy.stats as stats
from matplotlib import pyplot as plt
from IPython.core.pylabtools import figsize
import numpy as np
figsize(12.5, 4)
plt.rcParams['savefig.dpi'] = 300
plt.rcParams['figure.dpi'] = 300

import matplotlib.pyplot as plt
```

```
from mpl_toolkits.mplot3d import Axes3D

jet = plt.cm.jet

fig = plt.figure()
x = y = np.linspace(0, 5, 100)
X, Y = np.meshgrid(x, y)

plt.subplot(121)
uni_x = stats.uniform.pdf(x, loc=0, scale=5)
uni_y = stats.uniform.pdf(y, loc=0, scale=5)
M = np.dot(uni_x[:, None], uni_y[None, :])
im = plt.imshow(M, interpolation='none', origin='lower',
                cmap=jet, vmax=1, vmin=-.15, extent=(0, 5, 0, 5))

plt.xlim(0, 5)
plt.ylim(0, 5)
plt.title(u"Вид сверху на ландшафт,\nформируемый \
          равномерными априорными распределениями")

ax = fig.add_subplot(122, projection='3d')
ax.plot_surface(X, Y, M, cmap=plt.cm.jet, vmax=1, vmin=-.15)
ax.view_init(azim=390)
ax.set_xlabel('Значение $p_1$')
ax.set_ylabel('Значение $p_2$')
ax.set_zlabel('Плотность')
plt.title(u"Вид с другого ракурса на ландшафт,\nформируемый \
          равномерными априорными распределениями");
```

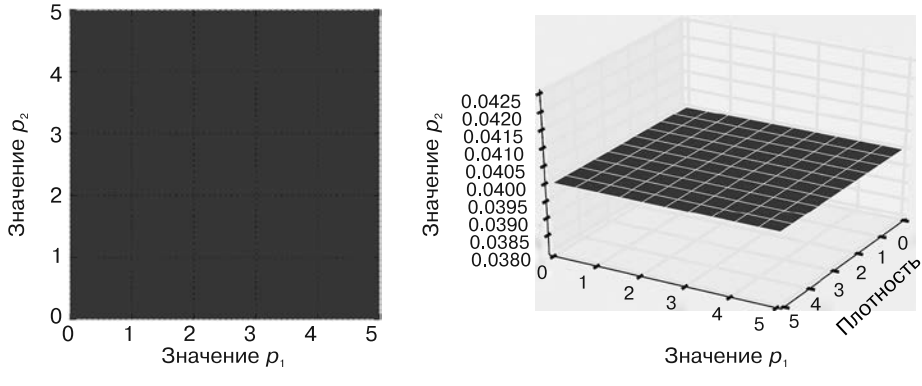

Рис. 3.1. *Слева:* вид сверху на ландшафт, формируемый равномерными априорными распределениями. *Справа:* вид с другого ракурса на ландшафт, формируемый равномерными априорными распределениями

Еще один вариант: если есть два априорных распределения — Exp(3) и Exp(10), то пространство охватывает все положительные числа на двумерной плоскости, а порождаемая априорными распределениями поверхность напоминает водопад, начинающийся в точке $(0, 0)$ и «проливающийся» на область положительных чисел.

Этот вариант визуализирован на рис. 3.2. Чем ближе цвет к темно-красному (на рис. 3.2 *слева внизу*), тем выше априорная вероятность для данной точки. И напротив, чем ближе цвет области к темно-синему (на рис. 3.2 *справа вверху*), тем ниже для нее априорные вероятности.

```
figsize(12.5, 5)
fig = plt.figure()
plt.subplot(121)

exp_x = stats.expon.pdf(x, scale=3)
exp_y = stats.expon.pdf(x, scale=10)
M = np.dot(exp_x[:, None], exp_y[None, :])
CS = plt.contour(X, Y, M)
im = plt.imshow(M, interpolation='none', origin='lower',
                cmap=jet, extent=(0, 5, 0, 5))
plt.title(u"Вид сверху на ландшафт,\nформируемый априорными\
           распределениями $Exp(3) и Exp(10)$")

ax = fig.add_subplot(122, projection='3d')
ax.plot_surface(X, Y, M, cmap=jet)
ax.view_init(azim=390)
ax.set_xlabel(u'Значение $p_1$')
ax.set_ylabel(u'Значение $p_2$')
ax.set_zlabel(u'Плотность')
plt.title(u"Вид с другого ракурса на ландшафт,\nформируемый априорными
          распределениями $Exp(3) и Exp(10)$");
```

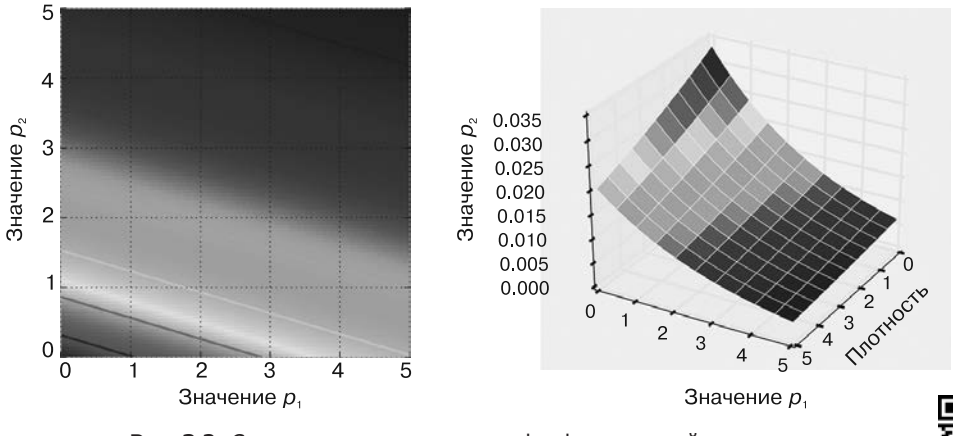

Рис. 3.2. *Слева:* вид сверху на ландшафт, формируемый априорными распределениями Exp(3) и Exp(10). *Справа:* вид с другого ракурса на ландшафт, формируемый априорными распределениями Exp(3) и Exp(10)

Это простые примеры в двумерном пространстве, поверхности в котором хорошо понятны нашему мозгу. На практике генерируемые на основе априорных распределений пространства и поверхности могут иметь гораздо более высокую размерность.

Но если эти поверхности описывают *априорные распределения* неизвестных величин, то что случится с пространством после того, как мы учтем наблюдаемые данные X? Данные X не поменяют само пространство, но изменят его поверхность посредством *растягивания структуры априорной поверхности*, чтобы отразить вероятные значения фактических параметров. Чем больше данных, тем больше растягивания, и исходная поверхность может измениться до неузнаваемости. А чем меньше данных, тем больше сохранится исходная форма поверхности. В любом случае получившаяся поверхность описывает новое *апостериорное распределение*.

Подчеркну еще раз: увы, визуализировать это все в пространстве большей размерности невозможно. В двумерном пространстве данные, по существу, поднимают исходную поверхность наверх, формируя своеобразные *высокие возвышенности*. Склонность исходных данных повышать апостериорную вероятность ограничивается априорным распределением вероятностей, и чем ниже априорная вероятность, тем сильнее противодействие. Следовательно, в предыдущем случае с двойным экспоненциальным распределением в углу (0, 0) может возникнуть возвышенность (или несколько возвышенностей), причем намного более высокая, чем те, которые возникнут ближе к точке (5, 5), поскольку возле точки (5, 5) противодействие сильнее (ниже априорная вероятность). Эта возвышенность отражает апостериорную вероятность того, в какой точке находятся фактические значения параметров. Важно отметить, что если априорная вероятность для точки равна 0, то отличной от 0 апостериорной вероятности для этой точки назначено не будет.

Пусть нам нужно выполнить вывод для двух пуассоновских распределений, каждое — с неизвестным параметром λ. Мы сравним результаты, воспользовавшись для неизвестных λ равномерным и экспоненциальным распределениями. Пусть дана наблюдаемая точка данных; визуализируем на рис. 3.3 ландшафты «до» и «после».

```
## Создаем данные наблюдений

# Размер выборки данных наблюдений; попробуйте различные значения
# этого параметра (не превышающие 100)
N = 1

# Фактические значения параметров, которые, конечно, мы не видим...
lambda_1_true = 1
lambda_2_true = 3

# ...а видим только сгенерированные на основе вышеупомянутых
# двух значений данные
data = np.concatenate([
    stats.poisson.rvs(lambda_1_true, size=(N, 1)),
    stats.poisson.rvs(lambda_2_true, size=(N, 1))
], axis=1)
print "наблюдаемые данные (двумерные,размер выборки = %d):" % N, data

# Подробности построения графика
x = y = np.linspace(.01, 5, 100)
```

```
likelihood_x = np.array([stats.poisson.pmf(data[:, 0], _x)
                         for _x in x]).prod(axis=1)
likelihood_y = np.array([stats.poisson.pmf(data[:, 1], _y)
                         for _y in y]).prod(axis=1)
L = np.dot(likelihood_x[:, None], likelihood_y[None, :])
```

```
[Output]:

наблюдаемые данные (двумерные,размер выборки = 1): [[0 6]]
```

```
figsize(12.5, 12)
# Внимание: matplotlib выполняет всю тяжелую работу!
plt.subplot(221)
uni_x = stats.uniform.pdf(x, loc=0, scale=5)
uni_y = stats.uniform.pdf(x, loc=0, scale=5)
M = np.dot(uni_x[:, None], uni_y[None, :])
im = plt.imshow(M, interpolation='none', origin='lower',
                cmap=jet, vmax=1, vmin=-.15, extent=(0, 5, 0, 5))
plt.scatter(lambda_2_true, lambda_1_true, c="k", s=50, edgecolor="none")
plt.xlim(0, 5)
plt.ylim(0, 5)
plt.title(u"Ландшафт, сформированный равномерными \naприорными \
          распределениями для $p_1, p_2$")

plt.subplot(223)
plt.contour(x, y, M * L)
im = plt.imshow(M * L, interpolation='none', origin='lower',
                cmap=jet, extent=(0, 5, 0, 5))
plt.title(u"Деформированный данными %d наблюдения ландшафт;\
          \nравномерные априорные распределения для $p_1, p_2$" % N)
plt.scatter(lambda_2_true, lambda_1_true, c="k", s=50, edgecolor="none")
plt.xlim(0, 5)
plt.ylim(0, 5)

plt.subplot(222)
exp_x = stats.expon.pdf(x, loc=0, scale=3)
exp_y = stats.expon.pdf(x, loc=0, scale=10)
M = np.dot(exp_x[:, None], exp_y[None, :])
plt.contour(x, y, M)
im = plt.imshow(M, interpolation='none', origin='lower',
cmap=jet, extent=(0, 5, 0, 5))
plt.scatter(lambda_2_true, lambda_1_true, c="k", s=50,
            edgecolor="none")
plt.xlim(0, 5)
plt.ylim(0, 5)
plt.title(u"Ландшафт, сформированный экспоненциальными\n\
          априорными распределениями для $p_1, p_2$")

plt.subplot(224)
# Апостериорная вероятность, равная мере правдоподобия,
# умноженной на априорную вероятность.
plt.contour(x, y, M * L)
```

```
im = plt.imshow(M * L, interpolation='none', origin='lower',
                cmap=jet, extent=(0, 5, 0, 5))

plt.scatter(lambda_2_true, lambda_1_true, c="k", s=50,
            edgecolor="none")
plt.title(u"Деформированный данными %d наблюдения ландшафт;\
          \nЭкспоненциальные априорные распределения для $p_1, p_2$" % N)
plt.xlim(0, 5)
plt.ylim(0, 5)
plt.xlabel(u'Значение $p_1$')
plt.ylabel(u'Значение $p_2$');
```

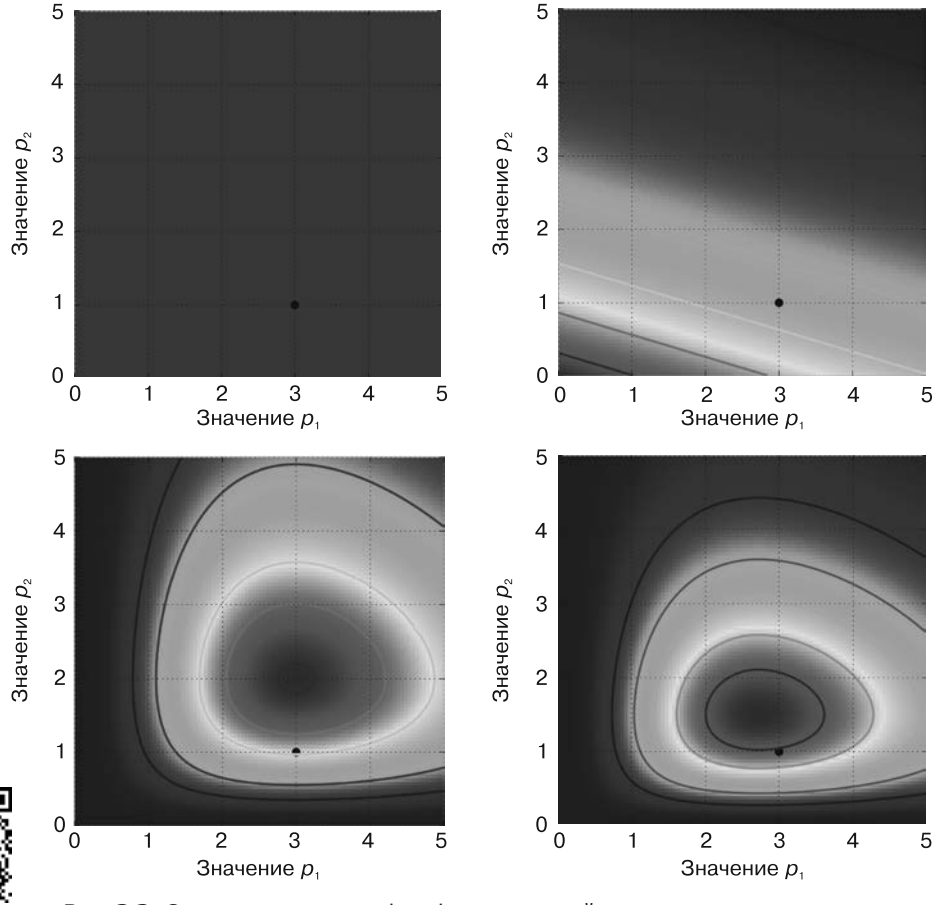

Рис. 3.3. *Слева вверху:* ландшафт, сформированный равномерными априорными распределениями для p_1, p_2. *Справа вверху:* ландшафт, сформированный экспоненциальными априорными распределениями для p_1, p_2. *Слева внизу:* деформированный данными наблюдения ландшафт при равномерных априорных распределениях для p_1, p_2. *Справа внизу:* деформированный данными наблюдения ландшафт при экспоненциальных априорных распределениях для p_1, p_2

Черная точка на каждом из четырех рисунков соответствует фактическим значениям параметров. График слева внизу представляет собой деформированный ландшафт с априорными распределениями Uniform(0, 5), а график справа внизу — деформированный ландшафт с экспоненциальными априорными распределениями. Обратите внимание, что апостериорные ландшафты различаются, хотя наблюдаемые данные в обоих случаях одинаковы. Причина в следующем: апостериорное распределение при экспоненциальном априорном (см. рис. 3.3, *справа внизу*) придает очень мало *апостериорного* веса значениям в правом верхнем углу графика, потому что *туда не вкладывает большого веса априорное распределение*. С другой стороны, равномерное априорное распределение вкладывает больше веса в правый верхний угол по сравнению с экспоненциальным, так что на него приходится и больший апостериорный вес.

Отмечу также, что самая верхняя точка, которой соответствует темно-красный цвет (на рисунке ближе к середине. — *Примеч. ред.*), смещена в случае экспоненциального априорного распределения к (0, 0), в результате чего экспоненциальное априорное распределение увеличивает вес в углу (0, 0). Даже при выборочной точке 1 возвышенности стремятся охватить фактическое значение параметра. Конечно, выполнение вывода при размере выборки 1 — исключительное наивное занятие, использование такого маленького размера выборки имеет смысл только для иллюстрации. Думаю, вам будет интересно попробовать различные размеры выборки (2; 5; 10; 100) и посмотреть, как будет меняться апостериорная возвышенность.

3.1.1. Изучаем ландшафт с помощью MCMC

Чтобы обнаружить апостериорную возвышенность, нужно проанализировать деформированное апостериорное пространство, полученное на основе априорной поверхности. Однако нет смысла наивно искать по всему пространству. Любой специалист в области компьютерных наук скажет вам, что сложность обхода *N*-мерного пространства экспоненциально растет с ростом *N*: размер пространства стремительно увеличивается при повышении *N* (см. статью в «Википедии» о проклятии размерности: https://ru.wikipedia.org/wiki/Проклятие_размерности). Есть ли у нас шанс найти эти скрытые возвышенности? В основе MCMC лежит идея интеллектуального поиска по пространству. Слово «поиск» подразумевает, что мы ищем конкретную точку — не вполне адекватное описание, ведь на самом деле мы ищем обширную возвышенность.

Напомню, что MCMC возвращает *выборки* из апостериорного распределения, а не само распределение. Если довести нашу аналогию с возвышенностями до предела, можно сказать, что MCMC как будто многократно задает вопрос: «Насколько вероятно, что этот найденный мной камешек родом с искомой возвышенности?» — и в завершение своей работы возвращает тысячи подходящих камешков в надежде восстановить из них исходную возвышенность. В терминологии MCMC и PyMC этими последовательно возвращаемыми «камешками» являются выборки, в совокупности носящие название *следов* (traces).

Говоря, что MCMC производит «интеллектуальный поиск», я на самом деле *надеюсь* на сходимость MCMC к областям высокой апостериорной вероятности. Для этого MCMC изучает близлежащие области и переходит в направлении более высокой вероятности. Под словом «сходимость» обычно подразумевают движение к одной точке пространства, но алгоритм MCMC движется к *широкой области* пространства и случайным образом проходит по ней, собирая выборки.

Зачем нужны тысячи выборок? На первый взгляд, возвращение пользователю тысяч выборок представляется неэффективным способом описания апостериорных распределений. Готов поспорить, что на самом деле этот способ как раз чрезвычайно эффективный. Рассмотрим альтернативные варианты.

1. Возвращение математической формулы для «цепей возвышенностей» потребовало бы описания многомерной поверхности с произвольными максимумами и минимумами. Это вовсе не просто.
2. Можно вернуть пик ландшафта (самую высокую точку возвышенности): это хотя и математически возможно и разумно (самая высокая точка соответствует наиболее вероятной оценке неизвестных величин), но не учитывает рельеф ландшафта, который очень важен для апостериорной уверенности в значениях неизвестных.

Помимо чисто вычислительных причин, вероятно, основной причиной возвращения выборок была возможность использования *закона больших чисел* для решения иным способом практически не решаемых задач. Обсуждение данной темы я отложу до главы 4. С помощью организации этих тысяч выборок в виде гистограммы можно воссоздать апостериорную поверхность.

3.1.2. Алгоритмы для MCMC

Существует крупное семейство алгоритмов для MCMC. Большинство из них можно описать, не вдаваясь в подробности, следующим образом.

1. Начать с текущего места.
2. Предложить перемещение на новое место (исследовать камешек неподалеку).
3. Принять/отвергнуть это новое место в зависимости от его соответствия данным и априорным распределениям (выяснить, не с искомой ли возвышенности этот камешек).
4. (А) Если предложение принимается, перейти на новое место. Вернуться к шагу 1.
   (Б) В противном случае не переходить на новое место. Вернуться к шагу 1.
5. После большого количества итераций вернуть все одобренные места.

Таким образом происходит перемещение в направлении областей с существующими апостериорными распределениями с эпизодическим сбором выборок по

пути. По достижении апостериорного распределения можно свободно собирать выборки, ведь они все, вероятно, относятся к апостериорному распределению.

Если текущая позиция алгоритма MCMC располагается в области чрезвычайно низкой вероятности, что зачастую случается в самом начале его работы (которая обычно начинается с произвольно выбранной точки в пространстве), алгоритм перейдет к точкам, которые, *вероятно, не относятся к апостериорному распределению*, но *лучше всех соседних точек*. Мы рассмотрим эту особенность позднее.

Обратите внимание, что в предыдущем псевдокоде для алгоритма имеет значение только текущая позиция (новые позиции исследуются лишь вблизи текущей). Это свойство можно описать как *отсутствие памяти*; то есть для алгоритма важно не то, *как* он попал в текущую точку, а только то, *что* он в ней находится.

3.1.3. Другие приближенные методы поиска апостериорных распределений

Помимо MCMC, существуют и другие процедуры, пригодные для вычисления апостериорных распределений. *Метод Лапласа* представляет собой способ приближенного вычисления апостериорного распределения с помощью простых функций. Более продвинутый — *вариационный байесовский метод* (variational Bayes). У каждого из этих трех методов — метода Лапласа, вариационного байесовского метода и обычного MCMC — есть свои достоинства и недостатки. Один мой приятель любит делить алгоритмы MCMC на две категории: «отстой» и «полный отстой». При этом конкретная разновидность алгоритмов MCMC, применяемых в PyMC, относится к категории «отстой».

3.1.4. Пример: кластеризация без учителя с использованием смеси распределений

Пусть у нас есть следующий набор данных:

```
figsize(12.5, 4)
data = np.loadtxt("data/mixture_data.csv", delimiter=",")

plt.hist(data, bins=20, color="k", histtype="stepfilled", alpha=0.8)
plt.title(u"Гистограмма набора данных")
plt.ylim([0, None])
plt.xlabel(u'Значение')
plt.ylabel(u'Количество')
print data[:10], "..."
```

```
[Output]:

[ 115.8568 152.2615 178.8745 162.935 107.0282 105.1914 118.3829
  125.377  102.8805 206.7133] ...
```

Что означают эти данные? Из рис. 3.4 видно, что форма данных бимодальная, то есть в них присутствуют два локальных максимума: один возле 120 и второй — возле 200. Вероятно, этот набор данных включает *два кластера*.

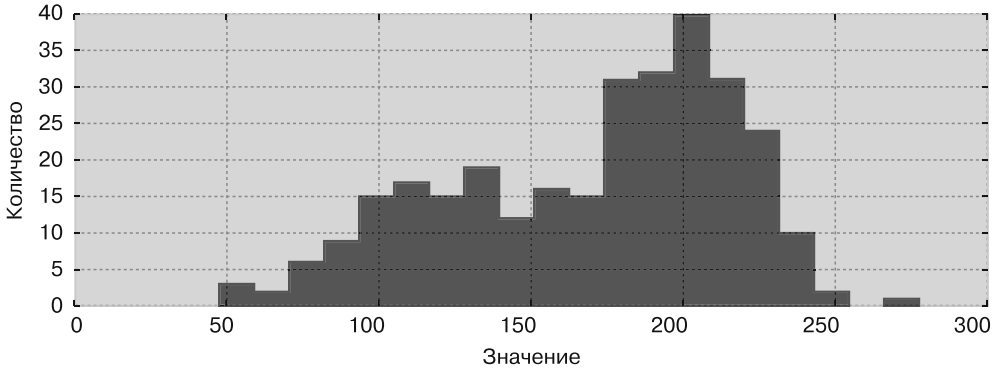

Рис. 3.4. Гистограмма набора данных

Этот набор данных может послужить хорошим примером для методики моделирования генерации данных из главы 2. Предположим, *как* могли быть созданы эти данные. Я предложил бы следующий алгоритм генерации данных.

1. Для каждой точки данных выбираем кластер 1 с вероятностью p, в противном случае выбираем кластер 2.
2. Берем случайную величину, распределенную по нормальному закону, с параметрами μ_i и σ_i, где i было выбрано на шаге 1.
3. Повторяем.

Этот алгоритм обеспечит такой же эффект, как наблюдается в нашем наборе данных, так что возьмем его за основу нашей модели. Конечно, p и параметры нормального распределения нам не известны. Следовательно, значения этих неизвестных величин нужно получить посредством вывода (обучения).

Обозначим нормальные распределения Nor_0 и Nor_1 (языку Python вообще свойственно начинать отсчет индексов переменных с 0). У обоих пока неизвестны средние значения (μ_i) и стандартные отклонения (σ_i), $i = 0{,}1$. Каждая конкретная точка данных может быть взята или из Nor_0, или из Nor_1, и мы предполагаем, что она относится к Nor_0 с вероятностью p. *Априори* нам неизвестна вероятность того, что точка относится к кластеру 1, так что промоделируем эту ситуацию с помощью переменной, равномерно распределенной по отрезку от 0 до 1. Назовем ее `p`.

Для распределения точек данных по кластерам имеет смысл воспользоваться стохастической переменной PyMC `Categorical`. Ее параметр представляет собой массив длины k вероятностей, которые в сумме должны давать 1, а атрибут `value` — выбранное случайным образом в соответствии с массивом вероятностей целочис-

ленное значение от 0 до $k - 1$ (в нашем случае $k = 2$). Таким образом, вносимый в переменную `Categorical` массив вероятностей имеет вид `[p, 1 – p]`.

```
import pymc as pm

p = pm.Uniform("p", 0., 1.)

assignment = pm.Categorical("assignment", [p, 1 - p], size=data.shape[0])
print "априорное распределение точек по кластерам при p = %.2f:" % p.value
print assignment.value[:10], "..."
```

```
[Output]:

априорное распределение точек по кластерам при p = 0.80:
[0 0 0 0 0 0 1 1 0 0] ...
```

Исходя из вида предыдущего набора данных, можно предположить, что стандартные отклонения двух его нормальных распределений различаются. Чтобы смоделировать наше незнание их стандартных отклонений, мы изначально задаем для них нормальные распределения на отрезке от 0 до 100. Фактически речь идет о τ — точности нормального распределения, но проще говорить в терминах стандартного отклонения.

Задача нашего кода PyMC состоит в преобразовании стандартного отклонения в точность с помощью такого соотношения:

$$\tau = \frac{1}{\sigma^2}.$$

В PyMC это можно сделать за один шаг, написав код:

```
taus = 1.0 / pm.Uniform("stds", 0, 100, size=2) ** 2
```

Обратите внимание, что мы указали `size=2`. Мы моделируем оба параметра τ с помощью одной переменной PyMC. Отмечу, что это не подразумевает наличия связи между ними, а сделано лишь ради большей лаконичности кода.

Необходимо также задать априорные распределения для центров кластеров. Эти центры фактически представляют собой параметры μ нормального распределения. Их априорные распределения можно смоделировать с помощью нормального распределения. По внешнему виду данных можно догадаться, где могут располагаться эти два центра — вероятно, неподалеку от 120 и 190 соответственно, хотя я не был бы слишком уверен в подобных оценках, сделанных навскидку. Поэтому мы зададим $\mu_0 = 120$, $\mu_1 = 190$ и $\sigma_{0,1} = 10$ (напомню, что мы указываем параметр τ, так что в переменную PyMC вносится значение $1 / \sigma^2 = 0{,}01$).

```
taus = 1.0 / pm.Uniform("stds", 0, 33, size=2) ** 2
centers = pm.Normal("centers", [120, 190], [0.01, 0.01], size=2)

"""
Следующие детерминистические функции присваивают значения 0 или 1
```

```
набору параметров, заключенных в массивах (1,2) "taus" и "centers".
"""

@pm.deterministic
def center_i(assignment=assignment, centers=centers):
    return centers[assignment]

@pm.deterministic
def tau_i(assignment=assignment, taus=taus):
    return taus[assignment]

print "Присвоенные случайные значения: ", assignment.value[:4], "..."
print "Назначенный центр: ", center_i.value[:4], "..."
print "Назначенная точность: ", tau_i.value[:4], "..."
```

```
[Output]:

Присвоенные случайные значения: [0 0 0 0] ...
Назначенный центр: [ 118.9889 118.9889 118.9889 118.9889] ...
Назначенная точность: [ 0.0041 0.0041 0.0041 0.0041] ...
```

```
# Объединяем с наблюдениями.
observations = pm.Normal("obs", center_i, tau_i, value=data,
                         observed=True)

# Создаем класс модели.
model = pm.Model([p, assignment, taus, centers])
```

В PyMC существует класс для MCMC — `MCMC` из основного пространства имен PyMC, который реализует алгоритм поиска MCMC. Для инициализации мы передадим ему экземпляр `Model`:

```
mcmc = pm.MCMC(model)
```

Метод, запускающий поиск MCMC по пространству, называется `pm.sample(iterations)`, где `iterations` — желаемое количество шагов алгоритма. Следующий код выполняет 50 000 шагов:

```
mcmc = pm.MCMC(model)
mcmc.sample(50000)
```

```
[Output]:

[-----------------100%-----------------] 50000 of 50000 complete in 31.5 sec
```

На рис. 3.5 мы строим график путей, называемых также *следами*, — неизвестных параметров (центры, точность и p) на текущий момент. Для получения следов можно воспользоваться методом `trace` созданного нами объекта `MCMC`, который принимает выбранное название переменной PyMC. Например, вызов

`mcmc.trace("centers")` извлекает объект `Trace`, из которого можно получить все следы путем обращения по индексу (с помощью синтаксиса `[:]` или `.gettrace()` или прихотливой индексации (fancy-indexing), например `[1000:]`).

```
figsize(12.5, 9)
plt.subplot(311)
line_width = 1
center_trace = mcmc.trace("centers")[:]

# Для отображения приятных глазу цветов далее в этой книге
colors = ["#348ABD", "#A60628"]
if center_trace[-1, 0] < center_trace[-1, 1]:
    colors = ["#A60628", "#348ABD"]

plt.plot(center_trace[:, 0], label=u"след центра 0",
         c=colors[0], lw=line_width)
plt.plot(center_trace[:, 1], label=u"след центра 1",
         c=colors[1], lw=line_width)
plt.title(u"Следы неизвестных параметров")
leg = plt.legend(loc="upper right")
leg.get_frame().set_alpha(0.7)

plt.subplot(312)
std_trace = mcmc.trace("stds")[:]
plt.plot(std_trace[:, 0], label=u"след стандартного отклонения кластера 0",
         c=colors[0], lw=line_width)
plt.plot(std_trace[:, 1], label=u"след стандартного отклонения кластера 1",
         c=colors[1], lw=line_width)
plt.legend(loc="upper left")

plt.subplot(313)
p_trace = mcmc.trace("p")[:]
plt.plot(p_trace, label=u"$p$: частота назначения точек в кластер 0",
         color="#467821", lw=line_width)
plt.xlabel(u"Шаги")
plt.ylim(0, 1)
plt.ylabel(u'Значение')
plt.legend();
```

Обратите внимание на следующие нюансы на рис. 3.5.

1. Следы сходятся не к отдельной точке, а к *распределению* вероятностей возможных точек. Это и есть *сходимость* в смысле алгоритма MCMC.
2. Выполнение вывода на основе первых нескольких тысяч точек — плохая идея, поскольку они не имеют никакого отношения к интересующему нас итоговому распределению. Следовательно, имеет смысл просто отбросить их, прежде чем использовать полученные выборки для вывода. Этот предшествующий сходимости период мы будем называть *периодом приработки* (burn-in period).

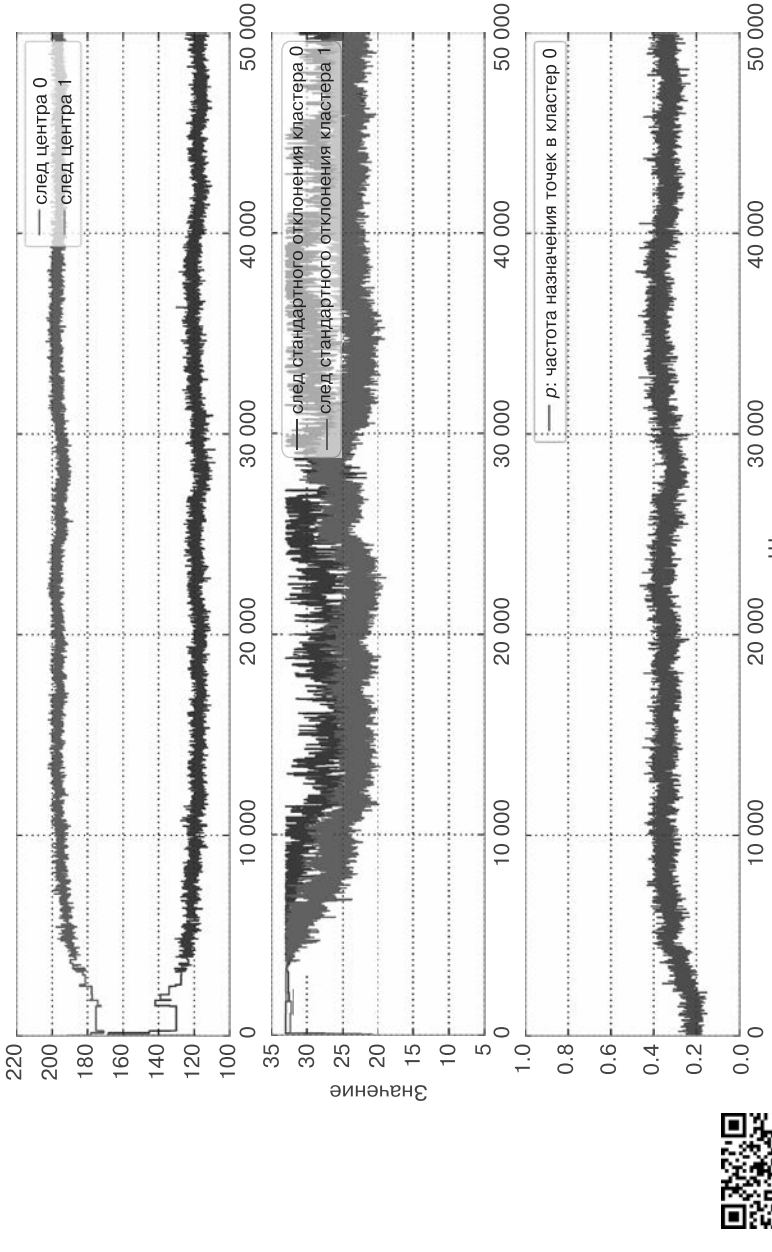

Рис. 3.5. Следы неизвестных параметров

3. Следы выглядят как случайный обход пространства, то есть демонстрируют связь каждой позиции с предыдущими. С одной стороны, это хорошо, с другой — не очень. Текущие позиции всегда будут связаны с предыдущими, но слишком жесткая подобная связь означает, что пространство не будет достаточно тщательно исследовано. Мы расскажем об этом подробнее в разделе, посвященном поиску проблем со сходимостью, далее в этой главе.

Для улучшения сходимости увеличим количество шагов MCMC. Запуск алгоритма MCMC заново после того, как он уже один раз был вызван, вовсе не означает запуска всего поиска сначала. Как видно из приведенного выше псевдокода алгоритма MCMC, значение имеет только текущая позиция (новые позиции исследуются неподалеку от текущей), сохраняемая неявным образом в атрибуте `value` переменных PyMC. Так что вполне допустимо прервать выполнение алгоритма MCMC, изучить полученные результаты, чтобы запустить его позднее. Атрибуты `value` не перезаписываются.

Произведем еще 100 000 выборок из MCMC и визуализируем ход выполнения (рис. 3.6).

```
mcmc.sample(100000)
```

```
[Output]:

[-----------------100%-----------------] 100000 of 100000 complete in 60.1 sec
```

```
figsize(12.5, 4)
center_trace = mcmc.trace("centers", chain=1)[:]
prev_center_trace = mcmc.trace("centers", chain=0)[:]

x = np.arange(50000)
plt.plot(x, prev_center_trace[:, 0], label=u"предыдущий след центра 0",
         lw=line_width, alpha=0.4, c=colors[1])
plt.plot(x, prev_center_trace[:, 1], label=u"предыдущий след центра 1",
         lw=line_width, alpha=0.4, c=colors[0])

x = np.arange(50000, 150000)
plt.plot(x, center_trace[:, 0], label=u"новый след центра 0",
         lw=line_width, c="#348ABD")
plt.plot(x, center_trace[:, 1], label=u"новый след центра 1",
         lw=line_width, c="#A60628")

plt.title(u"Следы неизвестных значений параметров центров \
          после еще 100 000 выборок")
leg = plt.legend(loc="upper right")
leg.get_frame().set_alpha(0.8)
plt.xlabel(u"Шаги")
plt.ylabel(u'Значение')
```

У метода `trace` из экземпляра `MCMC` есть именованный аргумент `chain`, с помощью которого можно указать, какой вызов `sample` нужно вернуть (часто приходится

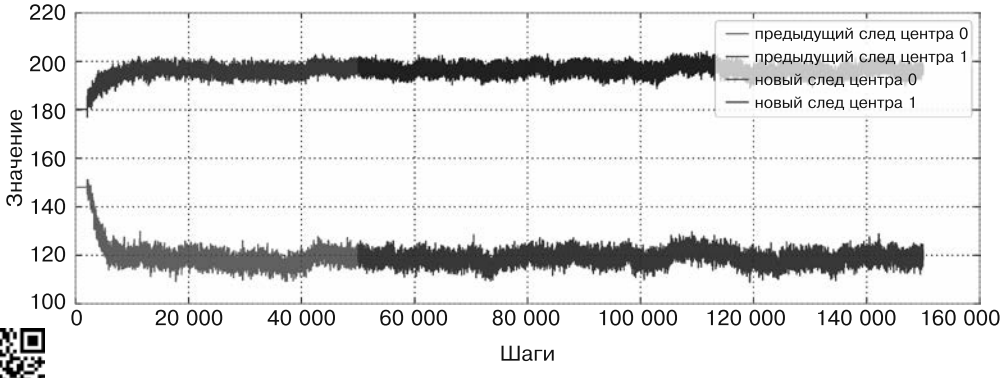

Рис. 3.6. Следы неизвестных значений параметров центров после еще 100 000 выборок

вызывать `sample` несколько раз, и возможность извлекать результаты предыдущих вызовов бывает очень востребована). Значение по умолчанию для аргумента `chain` равно `-1`, при этом возвращаются выборки из недавнего вызова `sample`.

Исследование кластеров. Мы не забыли, что основная наша задача состоит в определении кластеров. Мы уже определили апостериорные распределения для неизвестных величин. Построим апостериорные распределения переменных для центра и стандартного отклонения (рис. 3.7).

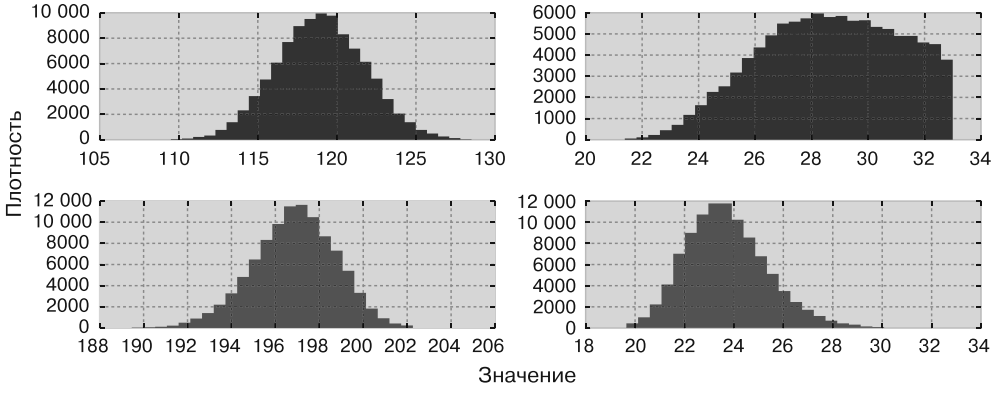

Рис. 3.7. *Слева сверху:* апостериорное распределение центра кластера 0. *Справа сверху:* апостериорное распределение стандартного отклонения кластера 0. *Слева снизу:* апостериорное распределение центра кластера 1. *Справа внизу:* апостериорное распределение стандартного отклонения кластера 1

```
figsize(11.0, 4)
std_trace = mcmc.trace("stds")[:]

_i = [1, 2, 3, 0]
```

```
for i in range(2):
    plt.subplot(2, 2, _i[2 * i])
    plt.title(u"Апостериорное распределение центра кластера %d" % i)
    plt.hist(center_trace[:, i], color=colors[i], bins=30,
             histtype="stepfilled")

    plt.subplot(2, 2, _i[2 * i + 1])
    plt.title(u"Апостериорное распределение стандартного отклонения кластера %d" % i)
    plt.hist(std_trace[:, i], color=colors[i], bins=30,
             histtype="stepfilled")
    plt.ylabel(u'Плотность')
    plt.xlabel(u'Значение')

plt.tight_layout();
```

Алгоритм MCMC пришел к выводу, что наиболее вероятные центры двух кластеров располагаются неподалеку от точек 120 и 200. Аналогичный вывод можно получить на основе стандартного отклонения.

Мы также получили апостериорные распределения для меток точек данных, их можно найти в `mcmc.trace("assignment")`. Графически они представлены на рис. 3.8. Ось *Y* отражает подвыборку апостериорных меток для каждой из точек данных. На оси *X* отложены отсортированные значения точек данных. Красный прямоугольник соответствует точкам, распределенным в кластер 1, а синий — в кластер 0.

```
import matplotlib as mpl
figsize(12.5, 4.5)
plt.cmap = mpl.colors.ListedColormap(colors)
plt.imshow(mcmc.trace("assignment")[::400, np.argsort(data)],
           cmap=plt.cmap, aspect=.4, alpha=.9)
plt.xticks(np.arange(0, data.shape[0], 40),
       ["%.2f" % s for s in np.sort(data)[::40]])
plt.ylabel(u"Апостериорная выборка")
plt.xlabel(u"Значение $i$-й точки данных")
plt.title(u"Апостериорные метки точек данных");
```

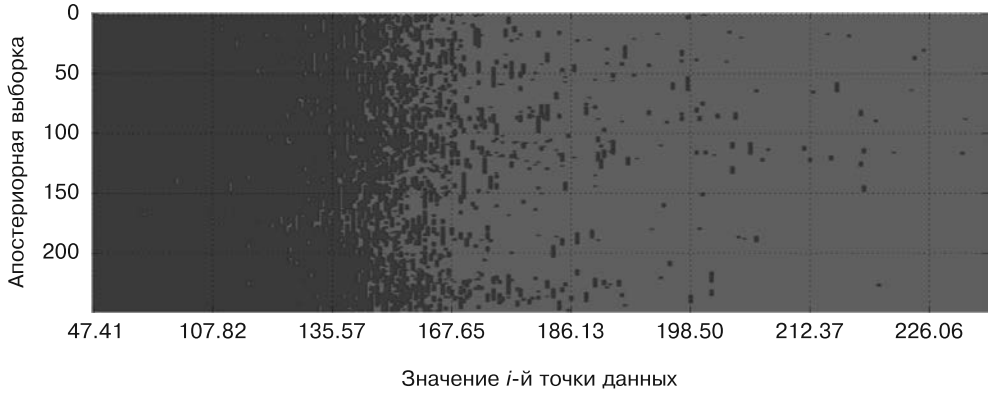

Рис. 3.8. Апостериорные метки точек данных

Судя по рис. 3.8, наибольшая неопределенность — в диапазоне от 150 до 170. Ситуация при этом предстает несколько в искаженном свете, поскольку ось X не отражает реальный масштаб (а отображает значение i-й отсортированной точки данных). Более понятная схема приведена на рис. 3.9, где оценивается *частота* принадлежности каждой из точек данных кластерам 0 и 1.

```
cmap = mpl.colors.LinearSegmentedColormap.from_list("BMH",colors)
assign_trace = mcmc.trace("assignment")[:]
plt.scatter(data, 1 - assign_trace.mean(axis=0), cmap=cmap,
            c=assign_trace.mean(axis=0), s=50)
plt.ylim(-0.05, 1.05)
plt.xlim(35, 300)
plt.title(u"Вероятность принадлежности точки данных кластеру 0")
plt.ylabel(u"Вероятность")
plt.xlabel(u"Значение точки данных");
```

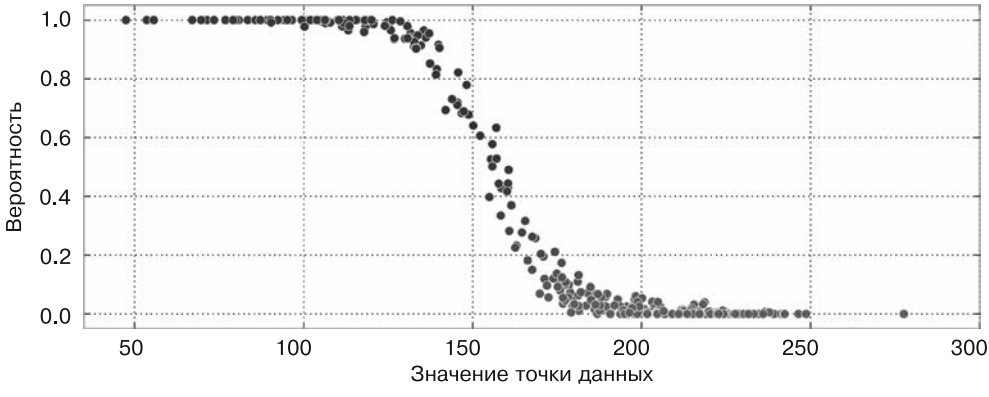

Рис. 3.9. Вероятность принадлежности точки данных кластеру 0

И хотя мы моделировали кластеры с помощью нормальных распределений, но получили в результате не просто одно нормальное распределение, *лучше всего* (что бы мы ни считали лучшим) подходящее к данным, а вероятностное распределение значений параметров нормального распределения. Как же нам выбрать одну-единственную пару значений для среднего и дисперсии и найти *как бы лучше всего подходящее* нормальное распределение?

Быстрый и некрасивый способ (теоретически обладающий весьма привлекательными свойствами, как мы увидим в главе 5) — воспользоваться *средним значением* апостериорных распределений. На рис. 3.10 мы совмещаем гауссовские функции плотности, используя в качестве параметров среднее апостериорных распределений при наших данных наблюдений.

```
norm = stats.norm
x = np.linspace(20, 300, 500)
posterior_center_means = center_trace.mean(axis=0)
```

```
posterior_std_means = std_trace.mean(axis=0)
posterior_p_mean = mcmc.trace("p")[:].mean()

plt.hist(data, bins=20, histtype="step", normed=True, color="k",
         lw=2, label=u"гистограмма данных")
y = posterior_p_mean * norm.pdf(x, loc=posterior_center_means[0],
                                scale=posterior_std_means[0])
plt.plot(x, y, label=u"кластер 0 (при средних апостериорных параметрах)", lw=3)
plt.fill_between(x, y, color=colors[1], alpha=0.3)

y = (1 - posterior_p_mean) * norm.pdf(x,
                                      loc=posterior_center_means[1],
                                      scale=posterior_std_means[1])
plt.plot(x, y, label=u"кластер 1 (при средних апостериорных параметрах)", lw=3)
plt.fill_between(x, y, color=colors[0], alpha=0.3)

plt.legend(loc="upper left")
plt.title(u"Визуализация кластеров при средних апостериорных параметрах");
```

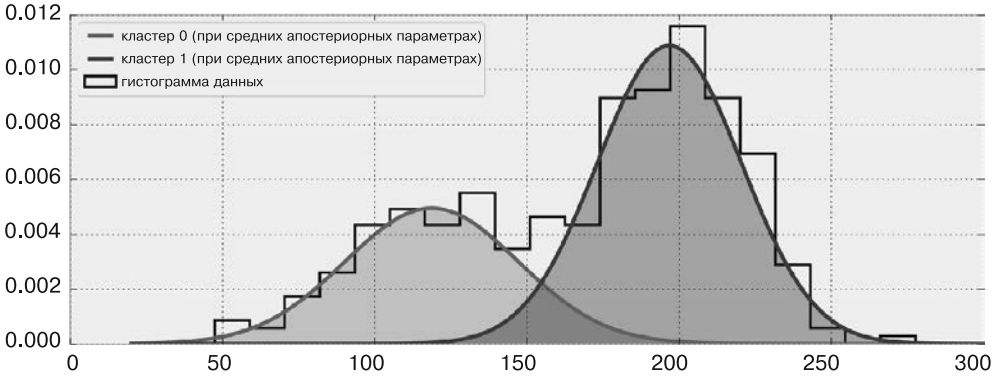

Рис. 3.10. Визуализация кластеров при средних апостериорных параметрах

3.1.5. Не смешивайте апостериорные выборки

Возможный (хотя и менее вероятный) сценарий на рис. 3.10 — очень большое стандартное отклонение кластера 0 и очень маленькое стандартное отклонение кластера 1. Такой вариант не противоречит данным наблюдений, хотя и хуже соответствует им, чем наш первоначальный вывод. Маленькое стандартное отклонение у *обоих* распределений чрезвычайно маловероятно, такая гипотеза вообще не подкрепляется данными. Следовательно, эти стандартные отклонения *зависят* друг от друга: если одно мало, то другое должно быть велико. На самом деле *все* наши неизвестные величины связаны подобным образом. Например, если стандартное отклонение велико, то диапазон возможных средних значений шире. И наоборот, малое стандартное отклонение ограничивает возможные средние значения очень

узким диапазоном. В результате выполнения MCMC возвращаются векторы, олицетворяющие выборки из неизвестных апостериорных распределений.

Элементы различных векторов нельзя использовать вместе — это нарушит вышеописанную логику. Скажем, выборка говорит, что у кластера 1 небольшое стандартное отклонение, следовательно, все остальные переменные в данной выборке должны это учитывать и иметь соответствующие значения. Впрочем, избежать подобной проблемы довольно легко: нужно просто правильно индексировать следы.

Еще один маленький пример. Пускай две переменные, x и y, связаны уравнением $x + y = 10$. На рис. 3.11 мы моделируем x как нормальную случайную переменную со средним значением 4 и анализируем 500 выборок.

```
import pymc as pm

x = pm.Normal("x", 4, 10)
y = pm.Lambda("y", lambda x=x: 10 - x, trace=True)

ex_mcmc = pm.MCMC(pm.Model([x, y]))
ex_mcmc.sample(500)

plt.plot(ex_mcmc.trace("x")[:])
plt.plot(ex_mcmc.trace("y")[:])
plt.xlabel(u'Шаги')
plt.ylabel(u'Значение')
plt.title(u"Демонстрация (предельного) случая зависимости \
          между неизвестными величинами");
```

```
[Output]:

[-----------------100%-----------------] 500 of 500 complete in 0.0 sec
```

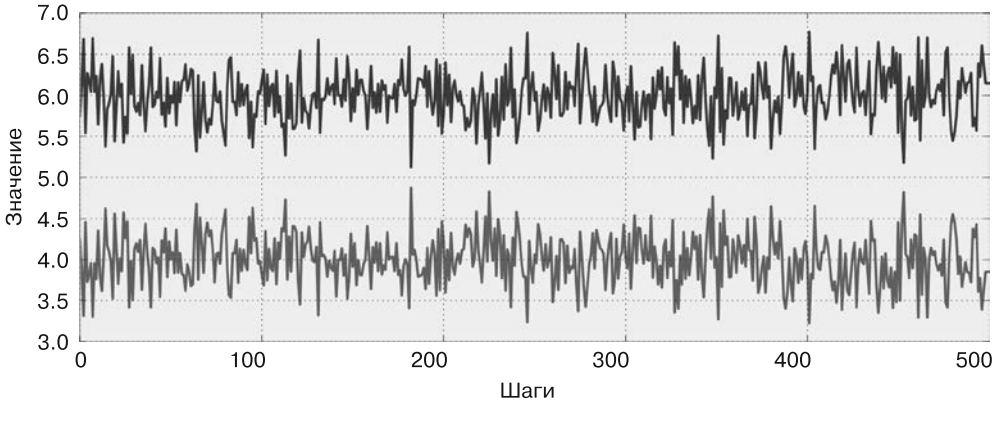

Рис. 3.11. Демонстрация (предельного) случая зависимости между неизвестными величинами

Как вы можете видеть, две переменные взаимосвязаны, так что складывать i-ю выборку величины x с j-й выборкой величины y было бы неправильно, за исключением случая $i = j$.

Возвращаемся к кластеризации: предсказание. Предыдущий пример кластеризации можно обобщить на k кластеров. Выбор частного случая $k = 2$ позволил нам лучше визуализировать MCMC, а также изучить несколько весьма интересных графиков.

Как же насчет предсказания? Пусть мы наблюдаем новую точку данных, скажем, $x = 175$, и хотели бы получить для нее метку кластера. Распределять ее просто в кластер, чей центр ближе всего, — довольно глупо, ведь такой способ не учитывает стандартные отклонения кластеров, а мы уже видели из предыдущих графиков, что они очень важны. На более формальном языке: нас интересует *вероятность* (поскольку полной уверенности в метках быть не может) того, что точка $x = 175$ относится к кластеру 1. Обозначим L_x (может принимать значение 0 или 1) то, что x относится к определенному кластеру. Нас интересует $P(L_x = 1 \mid x = 175)$.

«Наивный» метод вычисления этого значения: запустить предыдущий алгоритм MCMC еще раз, учтя дополнительную точку данных. Недостаток такого подхода в том, что вывод для каждой новой точки данных подобным образом будет слишком медленно выполняться. В качестве альтернативного варианта можно воспользоваться *менее точным*, но гораздо более быстрым методом.

Воспользуемся для этой цели теоремой Байеса. Как вы помните, она формулируется следующим образом:

$$P(A|X) = \frac{P(X|A)P(A)}{P(X)}.$$

В нашем случае A соответствует $L_x = 1$, а X — нашим данным наблюдений. Наблюдаем значение $x = 175$. Мы хотим выяснить при данном наборе параметров для нашего апостериорного распределения (μ_0, σ_0, μ_1, σ_1, p): «*Больше ли* вероятность того, что x относится к кластеру 1, чем вероятность того, что x относится к кластеру 0?» — где вероятность зависит от выбранных параметров.

$$P(L_x = 1 \mid x = 175) > P(L_x = 0 \mid x = 175)$$

$$\frac{P(x = 175 \mid L_x = 1)P(L_x = 1)}{P(x = 175)} > \frac{P(x = 175 \mid L_x = 0)P(L_x = 0)}{P(x = 175)}.$$

Поскольку знаменатели одинаковы, их можно проигнорировать — и тем лучше, ведь вычисление величины $P(x = 175)$ может оказаться непростой задачей.

$$P(x = 175 \mid L_x = 1)P(L_x = 1) > P(x = 175 \mid L_x = 0)P(L_x = 0).$$

```
norm_pdf = stats.norm.pdf
p_trace = mcmc.trace("p")[:]
```

```
x = 175
v = p_trace * norm_pdf(x, loc=center_trace[:, 0],
             scale=std_trace[:, 0]) > \
    (1 - p_trace) * norm_pdf(x, loc=center_trace[:, 1],
             scale=std_trace[:, 1])

print "Вероятность принадлежности кластеру 1:", v.mean()
```

```
[Output]:

Вероятность принадлежности кластеру 1: 0.025
```

Результат в виде вероятности вместо метки очень удобен. Вместо наивного:

```
L = 1 если вероятность > 0.5 в противном случае 0
```

можно оптимизировать гипотезы с помощью *функции потерь*. Этому посвящена вся глава 5.

3.1.6. Использование MAP для улучшения сходимости

Если вы запускали предыдущий пример, то могли заметить, что наши результаты не совсем одинаковы; ваше деление на кластеры могло оказаться более (или, наоборот, менее) рассредоточенным. Проблема в том, что следы представляют собой функции от *начальных значений* алгоритма MCMC.

Можно математически показать, что если алгоритм MCMC будет работать достаточно долго (в смысле количества шагов), то *забудет свою исходную позицию*. На самом деле именно это и понимается под сходимостью алгоритма MCMC (на практике, однако, полная сходимость недостижима). Следовательно, различные апостериорные результаты обычно означают, что MCMC еще не полностью сошелся и использовать выборки из него еще рано (период приработки нужно сделать более длительным).

На самом деле при плохих начальных значениях сходимость алгоритма может вообще отсутствовать или сильно замедлиться. В идеале желательно, чтобы марковская цепь начиналась в *локальном максимуме* ландшафта, ведь именно там существуют апостериорные распределения. Следовательно, если начать с пика ландшафта, можно избежать длительного периода приработки и некорректных выводов. Обычно такой пик называют *апостериорным максимумом* (maximum a posterior, MAP).

Конечно, нам неизвестна точка MAP. В PyMC есть объект для приближенной оценки или даже нахождения местоположения MAP. В основном пространстве имен PyMC есть объект `MAP`, принимающий экземпляр класса `Model` PyMC.

Установить переменные модели в их MAP-значения можно с помощью вызова метода `.fit()` из экземпляра `MAP`.

```
map_ = pm.MAP(model)
map_.fit()
```

Методы `MAP.fit()` достаточно гибки, и пользователь может выбрать метод оптимизации (в конце концов, речь идет о задаче оптимизации: мы ищем значения для максимизации ландшафта), ведь отнюдь не все алгоритмы оптимизации «рождены равными»[1]. По умолчанию при вызове `fit()` используется алгоритм оптимизации `fmin` из библиотеки SciPy (который пытается минимизировать *ландшафт со знаком «минус»*). В качестве альтернативы можно воспользоваться методом Пауэлла (Powell's method) — излюбленным методом пишущего на темы PyMC блогера Абрахама Флаксмена (Abraham Flaxman) [1] — путем вызова `fit(method='fmin powell')`. Я обычно использую метод по умолчанию, но если MCMC сходится слишком медленно или нет уверенности, что он вообще сойдется, то я начинаю экспериментировать с методом Пауэлла.

MAP можно также применять в качестве решения задачи вывода, поскольку математически это *наиболее вероятное* значение неизвестных. Но, как уже упоминалось в этой главе, выбор такого местоположения не учитывает неопределенность и не позволяет получить распределение.

Идея предварить вызов `mcmc` вызовом `MAP(model).fit()` обычно оказывается хорошей. Непосредственный вызов `fit()` не повлечет больших вычислительных расходов, но сэкономит время позднее за счет более короткого периода приработки.

Кстати, насчет периода приработки. Период приработки все равно не помешает, даже при использовании `MAP` перед вызовом `MCMC.sample`, просто для надежности. Указав параметр `burn` в вызове `sample`, можно сделать так, чтобы PyMC автоматически отбрасывал первые *n* выборок. А поскольку мы не знаем, когда марковская цепь полностью сойдется, я обычно отбрасываю первую *половину* (а иногда и до 90 %, при более длительной работе алгоритма) выборок. Если вернуться к примеру кластеризации, то новый код будет выглядеть примерно следующим образом:

```
model = pm.Model([p, assignment, taus, centers])

map_ = pm.MAP(model)
map_.fit() # сохраняем полученные в результате обучения
           # значения переменных в foo.value

mcmc = pm.MCMC(model)
mcmc.sample(100000, 50000)
```

[1] Аллюзия на знаменитую фразу Томаса Джефферсона: «All men are created equal» («Все люди рождены равными»), впервые упомянутую в Декларации независимости США.

3.2. Диагностика проблем со сходимостью

3.2.1. Автокорреляция

Автокорреляция (autocorrelation) — мера взаимосвязи ряда чисел с ним же самим. Равенство ее 1 означает абсолютную положительную автокорреляцию, 0 — отсутствие автокорреляции, а –1 — абсолютную отрицательную автокорреляцию. Если вы знакомы с понятием обычной *корреляции*, то представьте автокорреляцию всего лишь как корреляцию временного ряда $x\tau$ в момент t с ним же в момент $t - k$:

$$\mathrm{R}(k) = Corr(x_t, x_{t-k}).$$

Например, рассмотрим два временных ряда:

$$x_t \sim \mathrm{Normal}(0, 1), x_0 = 0,$$
$$y_t \sim \mathrm{Normal}(y_{t-1}, 1), y_0 = 0.$$

со следующими примерами путей:

```
figsize(12.5, 4)

import pymc as pm
x_t = pm.rnormal(0, 1, 200)
x_t[0] = 0
y_t = np.zeros(200)
for i in range(1, 200):
    y_t[i] = pm.rnormal(y_t[i - 1], 1)

plt.plot(y_t, label="$y_t$", lw=3)
plt.plot(x_t, label="$x_t$", lw=3)
plt.xlabel(u"Время, $t$")
plt.ylabel(u'Значение')
plt.title(u"Два различных ряда случайных величин")
plt.legend();
```

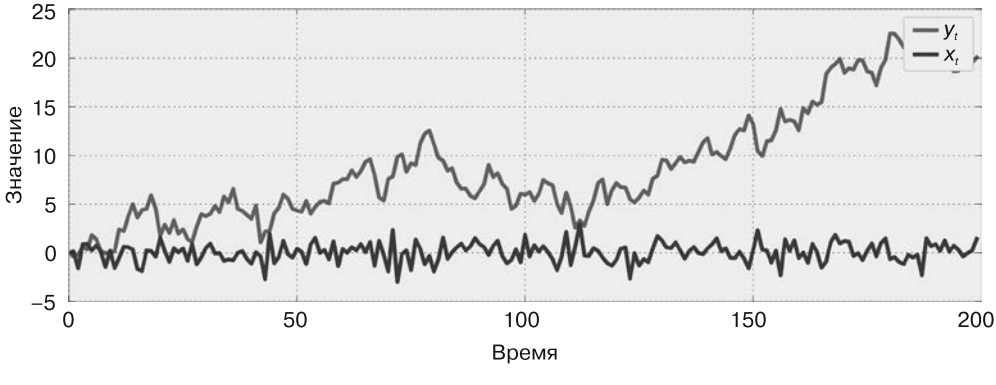

Рис. 3.12. Два различных ряда случайных величин

Автокорреляцию можно рассматривать как ответ на вопрос: «Поможет ли известное значение ряда на момент *s* узнать значение на момент *t*?» В случае ряда x_t ответ на этот вопрос отрицательный. По построению x_t представляют собой независимые случайные переменные. Если я скажу вам, что $x_2 = 0{,}5$, поможет ли вам это точнее сказать, чему равно x_3? Нет.

С другой стороны, ряд y_t автокоррелирован. По построению при известном $y_2 = 10$ можно с уверенностью сказать, что значение y_3 также близко к 10. Можно даже сделать (с меньшей долей уверенности) предположение о значении y_4: вероятно, оно не будет близко к 0 или 20, но значение 5 вполне возможно. Можно высказать аналогичное суждение и относительно y_5, но еще менее уверенно. Логическое следствие вышеизложенного: по мере роста k (сдвига между временны*м*и точками) автокорреляция уменьшается. Это отображено на рис. 3.13. Красный ряд *(внизу)* представляет собой белый шум (автокорреляция отсутствует), а синий ряд *(вверху)* рекурсивен (с очень сильной автокорреляцией).

```
def autocorr(x):
    # Взято из http://tinyurl.com/afz57c4
    result = np.correlate(x, x, mode='full')
    result = result / np.max(result)
    return result[result.size / 2:]

colors = ["#348ABD", "#A60628", "#7A68A6"]
x = np.arange(1, 200)
plt.bar(x, autocorr(y_t)[1:], width=1, label="$y_t$",
        edgecolor=colors[0], color=colors[0])
plt.bar(x, autocorr(x_t)[1:], width=1, label="$x_t$",
        color=colors[1], edgecolor=colors[1])

plt.legend(title=u"автокорреляция")
plt.ylabel(u"Измеренная корреляция \nмежду $y_t$ и $y_{t-k}$.")
plt.xlabel("$k$ (сдвиг)")
plt.title(u"График автокорреляции $y_t$ и $x_t$ при различных\
          сдвигах $k$");
```

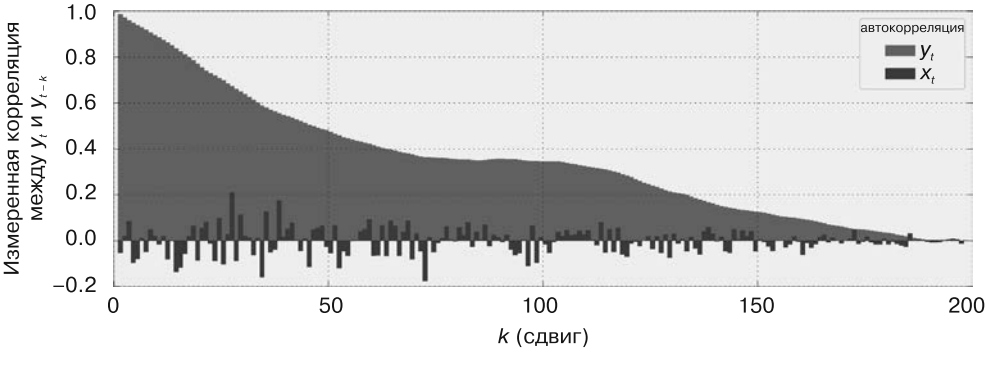

Рис. 3.13. График автокорреляции y_t и x_t при различных сдвигах k

Обратите внимание, что на рис. 3.13 по мере роста k уменьшается автокорреляция y_t, начиная с очень высоких значений. Сравнивая это с автокорреляцией x_t, напоминающей шум (чем на самом деле она и является), можно прийти к заключению об отсутствии автокорреляции в ряде x_t.

Какое отношение это имеет к сходимости MCMC? Сущность алгоритма MCMC такова, что всегда возвращаются выборки с автокорреляцией (вследствие того, что алгоритм «обходит» пространство: перемещается из текущей позиции в близлежащую).

Цепь Маркова, хорошо анализирующая пространство, будет демонстрировать очень высокую автокорреляцию. Образно можно сказать, что если след извивается, подобно руслу реки, и не стабилизируется, то автокорреляция будет высока. Как вариант, приведу мое определение извилистой реки.

Представьте себе, что вы молекула воды в реке на рис. 3.14. Если я знаю, в каком месте реки вы находитесь, то могу предположить с неплохой вероятностью, где вы окажетесь далее. С другой стороны, говорят, что цепь «хорошо перемешивает», если у нее низкая автокорреляция. «Хорошо перемешивает» — не просто слова. В идеале ваша цепь должна вести себя так, как река на рис. 3.15. В этом случае я практически не имею понятия, где вы можете оказаться в следующий момент. Вот такое «хорошее перемешивание» и означает низкую автокорреляцию.

Рис. 3.14. Извилистая река [2]

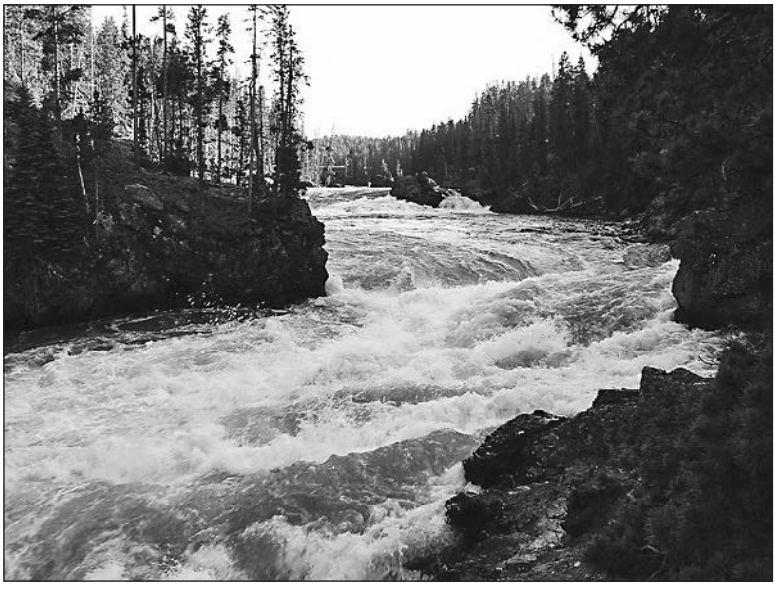

Рис. 3.15. Бурная, хорошо перемешивающая река [3]

3.2.2. Прореживание

При высокой автокорреляции между апостериорными выборками возникает еще одна проблема. Многие алгоритмы постобработки требуют *независимости* выборок друг от друга. Эту проблему можно решить или по крайней мере сгладить, возвращая пользователю лишь каждую n-ю выборку, благодаря чему автокорреляция снижается. На рис. 3.16 мы построили графики автокорреляции для y_t при различном уровне прореживания:

```
max_x = 200 / 3 + 1
x = np.arange(1, max_x)

plt.bar(x, autocorr(y_t)[1:max_x], edgecolor=colors[0],
        label=u"без прореживания", color=colors[0], width=1)
plt.bar(x, autocorr(y_t[::2])[1:max_x], edgecolor=colors[1],
        label=u"оставляем каждую 2-ю выборку", color=colors[1],width=1)
plt.bar(x, autocorr(y_t[::3])[1:max_x], width=1, edgecolor=colors[2],
        label=u"оставляем каждую 3-ю выборку", color=colors[2])

plt.autoscale(tight=True)
plt.legend(title=u"График автокорреляции для $y_t$", loc="lower left")
plt.ylabel(u"Оценка корреляции \nмежду $y_t$ и $y_{t-k}$.")
plt.xlabel(u"$k$ (сдвиг)")
plt.title(u"Автокорреляция $y_t$ (без прореживания и с прореживанием) \
          при различных сдвигах $k$");
```

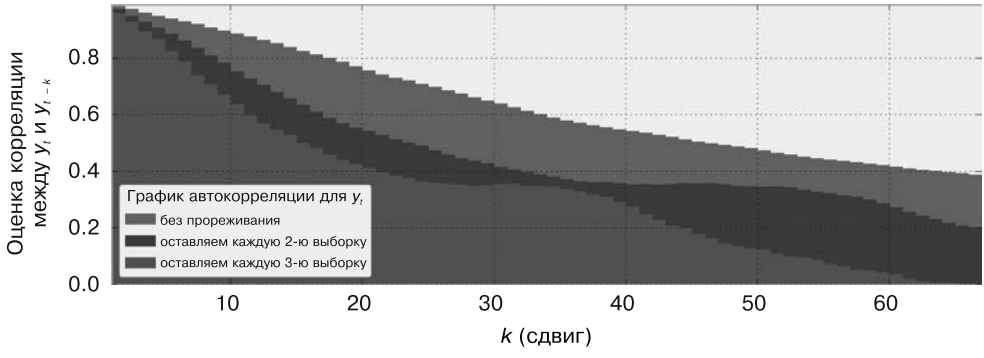

Рис. 3.16. График автокорреляции y_t (без прореживания и с прореживанием) при различных сдвигах k

При более сильном прореживании автокорреляция уменьшается быстрее. Однако у этого есть и обратная сторона: более активное прореживание требует большего количества итераций MCMC для возврата того же числа выборок. Например, 10 000 выборок без прореживания эквивалентны 100 000 выборок при прореживании уровня 10 (хотя автокорреляция в более поздних выборках может оказаться слабее).

Какой уровень прореживания оптимален? Определенная автокорреляция всегда присутствует в возвращаемых выборках, вне зависимости от степени прореживания. Так что если автокорреляция быстро стремится к 0, то, скорее всего, все в порядке. Обычно прореживание более чем 10-го уровня не требуется.

При вызове метода `sample` доступен параметр `thinning`, например:

```
sample(10000, burn = 5000, thinning = 5)
```

3.2.3. Функция pymc.Matplot.plot()

Создание вручную гистограмм, графиков автокорреляций и графиков следов при каждом выполнении MCMC — излишний труд. Создатели PyMC предусмотрели отдельную утилиту визуализации специально для этой цели.

Как понятно из названия, модуль `pymc.Matplot.plot()` содержит не слишком удачно названную функцию `plot`. Я предпочитаю импортировать ее под именем `mcplot`, чтобы не возникало конфликтов с другими пространствами имен. Функция `plot` (или `mcplot`) принимает на входе объект `MCMC` и возвращает апостериорные распределения, следы и автокорреляции для всех переменных (до десяти переменных).

Построим с помощью этой утилиты график центров кластеров после 25 000 дополнительных выборок при `thinning` = 10 (рис. 3.17).

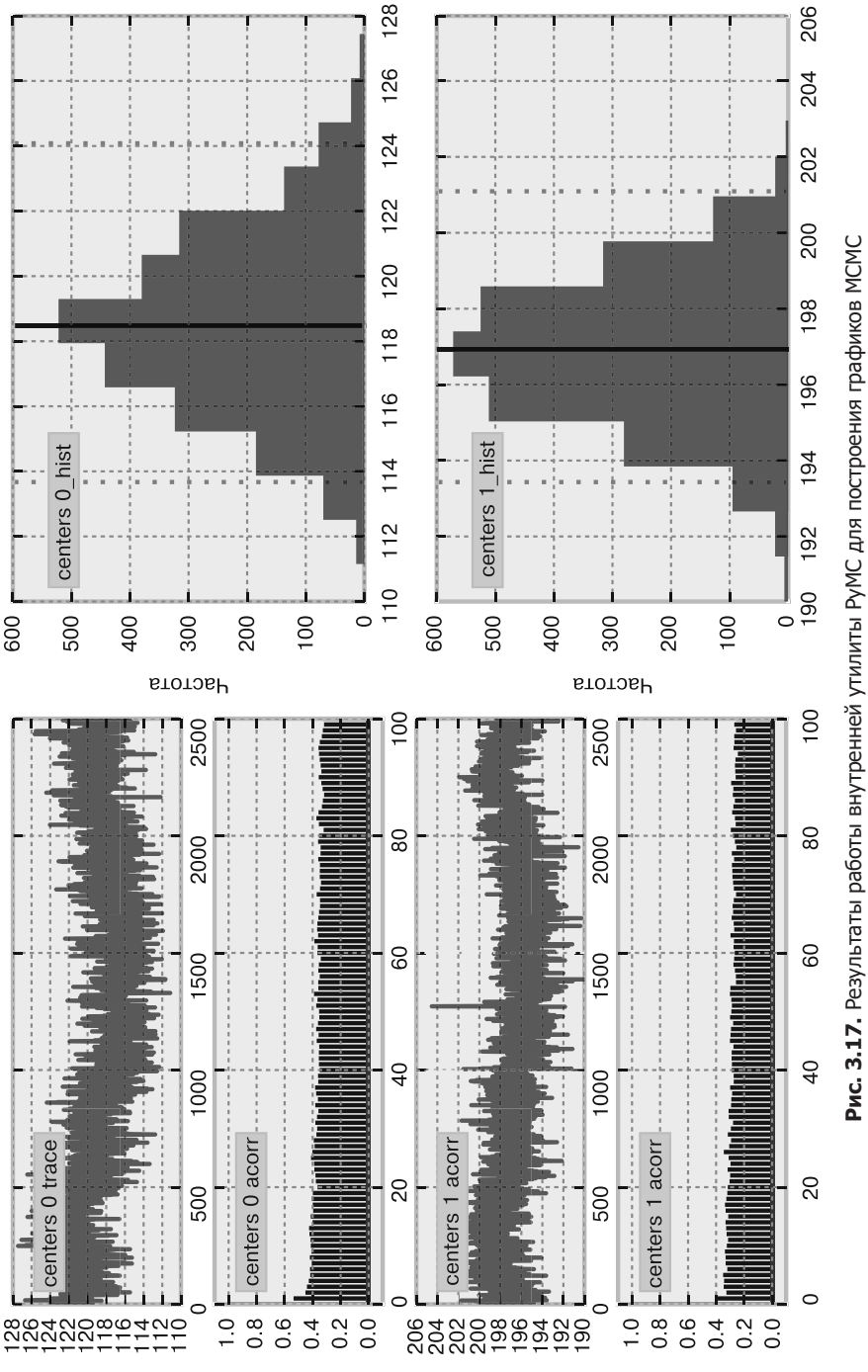

Рис. 3.17. Результаты работы внутренней утилиты PyMC для построения графиков MCMC

```
from pymc.Matplot import plot as mcplot

mcmc.sample(25000, 0, 10)
mcplot(mcmc.trace("centers", 2), common_scale=False)
```

```
[Output]:

[-----------------100%-----------------] 25000 of 25000 complete in 16.1 sec
Plotting centers_0
Plotting centers_1
```

На рис. 3.17 приведены на самом деле два графика: по одному для каждой неизвестной величины в переменной `centers`. На каждом из этих графиков на подграфике слева вверху отображен след переменной. Он удобен для анализа возможной «извилистости», возникающей при отсутствии сходимости.

Крупные графики справа представляют собой гистограммы выборок с некоторой дополнительной информацией. Жирная вертикальная линия соответствует апостериорному среднему значению — удобному сводному показателю апостериорного распределения. Интервал между двумя вертикальными штриховыми линиями для каждого из апостериорных распределений отражает 95%-ный байесовский доверительный интервал (не путайте с просто 95%-ным доверительным интервалом). Обсуждать последний я не стану, а вот первый из них можно интерпретировать как «существует 95%-ная вероятность попадания интересующего нас параметра в этот интервал» (можно поменять используемые по умолчанию значения параметров и получить другие интервалы, не 95%-ные).

Чрезвычайно важно указывать этот интервал, когда вы сообщаете другим людям полученные результаты. Одна из наших важнейших целей при изучении байесовских методов — ясно понимать степень неопределенности значений неизвестных. В сочетании с апостериорным средним значением 95%-ный байесовский доверительный интервал дает возможность понятным образом обмениваться информацией о вероятном расположении неизвестных (благодаря среднему значению) и степени неопределенности (отражаемой шириной интервала).

3.3. Полезные советы по поводу MCMC

Байесовский вывод был бы эталонным методом, если бы не вычислительные сложности MCMC. На самом деле именно MCMC отбивает у большинства пользователей охоту к применению байесовского вывода на практике. В этом разделе я расскажу про несколько удачных эвристических алгоритмов улучшения сходимости и ускорения работы движка MCMC.

3.3.1. Интеллектуальный выбор начальных значений

Было бы неплохо начать алгоритм MCMC с точки, расположенной неподалеку от апостериорного распределения, чтобы как можно быстрее приступить к взятию правильных выборок. Можно помочь алгоритму, указав, где, *по нашему мнению*, будет располагаться апостериорное распределение, с помощью параметра `value` при создании стохастической переменной. Во многих случаях можно высказать разумную гипотезу относительно значения этого параметра. Например, если наши данные взяты из нормального распределения и нам требуется оценить параметр μ, то разумно будет начать со *среднего значения* данных:

```
mu = pm.Uniform( "mu", 0, 100, value = data.mean() )
```

Большинство параметров моделей можно оценить с точки зрения частотного подхода. Эти оценки — неплохие начальные значения для алгоритмов MCMC. Конечно, для некоторых переменных это не всегда удается, но всегда лучше включить как можно больше подходящих значений. Даже если гипотезы окажутся ошибочными, MCMC все равно сойдется к должному распределению, так что вы ничем не рискуете. Именно это и делает метод `MAP`, предоставляя MCMC хорошие начальные значения. Так зачем же тратить силы на задание пользовательских значений? Дело в том, что передача методу `MAP` хороших значений поможет ему найти апостериорный максимум. Немаловажно также, что *плохие начальные значения* — источник основных проблем PyMC, они могут отрицательно повлиять на сходимость.

3.3.2. Априорные распределения

При плохо выбранных априорных распределениях алгоритм MCMC может не сойтись или по крайней мере испытывать проблемы со сходимостью. Задумайтесь, что будет, если выбранное априорное распределение вообще не включает фактическое значение параметра. Априорная вероятность неизвестной величины будет равна 0, а значит, и апостериорная тоже. Это приведет к некорректным результатам.

Поэтому лучше всего осмотрительно выбирать априорные распределения. Зачастую отсутствие сходимости или скопление выборок возле границ означает, что с выбранными априорными распределениями что-то не так (см. подраздел 3.3.3).

3.3.3. Народная теорема статистических расчетов

Народная теорема, если этот термин вам незнаком, — неписаная истина, известная всем работающим в определенной сфере. Народная теорема байесовских вычислений формулируется следующим образом:

> *если вы столкнулись с вычислительными трудностями, то, вероятно, с вашей моделью что-то не в порядке.*

3.4. Выводы

PyMC предоставляет отличную прикладную часть для байесовского вывода в основном благодаря абстрагированию внутренних механизмов работы MCMC от пользователя. Несмотря на это, необходимо приложить дополнительные усилия, чтобы не допустить искажения вывода за счет итеративной природы MCMC.

Другие библиотеки MCMC, например Stan, применяют для MCMC другие алгоритмы, основанные на современных научных исследованиях в данной сфере. Эти алгоритмы менее подвержены проблемам со сходимостью, а следовательно, еще более абстрагируют MCMC от пользователя.

В главе 4 мы изучим важнейшую, с моей точки зрения (частично вследствие ее практической полезности, а частично — потому, что ее часто применяют неправильно), теорему: *закон больших чисел*.

3.5. Библиография

1. *Flaxman A.* Powell's Methods for Maximization in PyMC // Healthy Algorithms, 28, Feb 2013. http://healthyalgorithms.com/2012/02/09/powells-method-for-maximization-in-pymc/.
2. Из Meandering river blue. The-LizardQueen. https://flic.kr/p/95jKe.
3. Из Close up to lower water falls... Tim Pearce. https://flic.kr/p/92V9ip.

4 Величайшая из несформулированных теорем

4.1. Введение

Эта глава посвящена идее, регулярно всплывающей в наших мыслях, но редко формулируемой явным образом за пределами книг, посвященных статистике. Фактически эта простая идея использовалась буквально в каждом приведенном выше примере.

4.2. Закон больших чисел

Пусть Z_i — N независимых выборок из какого-то распределения вероятностей. Согласно закону больших чисел справедливо следующее (если математическое ожидание $E[Z]$ не равно бесконечности):

$$\frac{1}{N}\sum_{i=1}^{N} Z_i \to E[Z], N \to \infty.$$

Другими словами:

> среднее значение набора случайных переменных, взятых из одного распределения, сходится к математическому ожиданию этого распределения.

Может показаться, что это довольно скучный результат, но на деле он станет полезнейшим из всех ваших инструментов.

4.2.1. Интуиция

Если вышеприведенный закон оказался в чем-то для вас неожиданным, то прояснить его поможет следующий простой пример.

Рассмотрим случайную переменную Z, которая может принимать только два значения: c_1 и c_2. Пусть у нас имеется большое количество выборок Z, где отдельные выборки обозначены Z_i. Закон утверждает, что можно аппроксимировать

математическое ожидание Z посредством усреднения по всем выборкам. Рассмотрим среднее значение:

$$\frac{1}{N}\sum_{i=1}^{N} Z_i$$

По построению Z_i может принимать только значения c_1 и c_2, так что можно разбить общую сумму на две составляющие:

$$\frac{1}{N}\sum_{i=1}^{N} Z_i = \frac{1}{N}\left(\sum_{Z_i=c_1} c_1 + \sum_{Z_i=c2} c_2\right) =$$

$$= c_1 \sum_{Z_i=c_1} \frac{1}{N} + c_2 \sum_{Z_i=c_2} \frac{1}{N} =$$

$$= c_1 \times (\text{приближенная частота, с которой встречается } c_1) +$$

$$+ c_2 \times (\text{приближенная частота, с которой встречается } c_2) \approx$$

$$\approx c_1 \times P(Z = c_1) + c_2 \times P(Z = c_2) = E[Z].$$

Точное равенство достижимо только в пределе, но к нему можно все более приближаться благодаря использованию в среднем значении все большего количества выборок. Закон больших чисел справедлив практически *для всех распределений*, за исключением незначительного количества важных частных случаев.

4.2.2. Пример: сходимость пуассоновских случайных переменных

На рис. 4.1 приведен график, демонстрирующий закон больших чисел в действии для трех различных последовательностей пуассоновских случайных переменных.

Мы производим выборку `sample_size = 100000` пуассоновских случайных переменных с параметром $\lambda = 4{,}5$ (напоминаю, что математическое ожидание пуассоновской случайной переменной равно значению ее параметра). Вычислим среднее значение для первых n выборок, для n от 1 до `sample_size`.

```
%matplotlib inline
import numpy as np
from IPython.core.pylabtools import figsize
import matplotlib.pyplot as plt
plt.rcParams['savefig.dpi'] = 300
plt.rcParams['figure.dpi'] = 300

figsize(12.5, 5)
import pymc as pm

sample_size = 100000
expected_value = lambda_ = 4.5
poi = pm.rpoisson
```

```
N_samples = range(1,sample_size,100)

for k in range(3):
    samples = poi(lambda_, size = sample_size)
    partial_average = [samples[:i].mean() for i in N_samples]

    plt.plot(N_samples, partial_average, lw=1.5,label=u"среднее значение \
             по $n$ выборкам; посл. %d"%k)

plt.plot(N_samples, expected_value*np.ones_like(partial_average),
         ls="--", label=u"фактическое математическое ожидание", c="k")

plt.ylim(4.35, 4.65)
plt.title(u"Сходимость среднего значения случайных \n переменных к их \
          математическому ожиданию")
plt.ylabel(u"Среднее по $n$ выборкам")
plt.xlabel(u"Число выборок, $n$")
plt.legend();
```

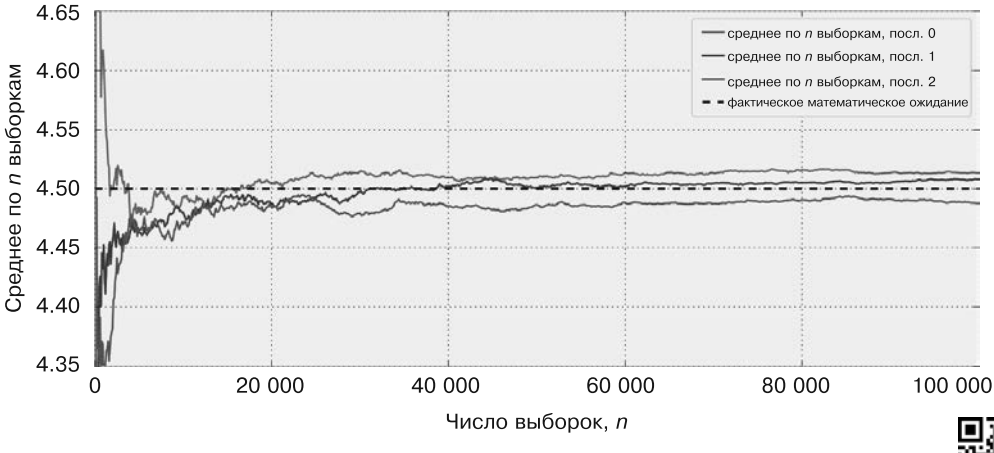

Рис. 4.1. Сходимость среднего значения случайных переменных к их математическому ожиданию

Из рис. 4.1 понятно, что при маленьком размере выборки разброс средних значений сильнее (посмотрите, как в конце выравнивается график среднего значения, зазубренный и скачущий в начале). Все три кривые приближаются к значению 4,5, но лишь «заигрывают» с ним по мере роста *n*. Математики и специалисты по статистике называют такое «заигрывание» *сходимостью*.

Еще один вполне уместный вопрос: а как быстро алгоритм сходится к математическому ожиданию? Давайте построим какой-нибудь новый график. Для конкретного *n* произведем несколько тысяч испытаний и вычислим, насколько далеко в среднем мы от фактического математического ожидания. Но постойте, что значит

«вычислить в среднем»? Это просто опять закон больших чисел! Например, пусть нас интересует для конкретного n величина:

$$D(n) = \sqrt{E\left[\left(\frac{1}{n}\sum_{i=1}^{n} Z_i - 4,5\right)^2\right]}.$$

Эта формула отражает расстояние от фактического значения (в среднем) для некоторого n (благодаря корню квадратному размерности этой величины и наших случайных переменных одинаковы). А математическое ожидание можно приближенно вычислить с помощью закона больших чисел: вместо усреднения Z_i можно вычислить много раз следующие значения и усреднить их:

$$Y_{n,k} = \left(\frac{1}{n}\sum_{i=1}^{n} Z_i - 4,5\right)^2.$$

В результате многократного вычисления $Y_{n,k}$ всякий раз с новыми Z_i и усреднения получаем:

$$\frac{1}{N}\sum_{k=1}^{N} Y_{n,k} \to E[Y_n] = E\left[\left(\frac{1}{n}\sum_{i=1}^{n} Z_i - 4,5\right)^2\right].$$

Наконец, извлекаем квадратный корень:

$$\sqrt{\frac{1}{N}\sum_{k=1}^{N} Y_{n,k}} \approx D(n).$$

```
figsize(12.5, 4)

N_Y = 250 # Такое количество выборок будет использовано
          # для приближенного вычисления дисперсии D(N).
N_array = np.arange(1000, 50000, 2500)
D_N_results = np.zeros(len(N_array))

lambda_ = 4.5
expected_value = lambda_ # При X ~ Poi(lambda), E[X] = lambda

def D_N(n):
    """
    Эта функция приближенно вычисляет D_n — среднюю дисперсию
    при выборке размера n.
    """
    Z = poi(lambda_, size = (n, N_Y))
    average_Z = Z.mean(axis=0)
    return np.sqrt(((average_Z - expected_value)**2).mean())

for i,n in enumerate(N_array):
    D_N_results[i] = D_N(n)
plt.xlabel("$N$")
plt.ylabel(u"Ожидаемое среднеквадратическое расстояние от фактического значения")
```

```
plt.plot(N_array, D_N_results, lw=3,
label=u"ожидаемое расстояние между \n математическим ожиданием и\
       \nсредним значением по $N$ случайных переменных")
plt.plot(N_array, np.sqrt(expected_value)/np.sqrt(N_array), lw=2,
         ls="--", label=r"$\frac{\sqrt{\lambda}}{\sqrt{N}}$")
plt.legend()
plt.title(u"Насколько быстро сходится среднее по выборкам?");
```

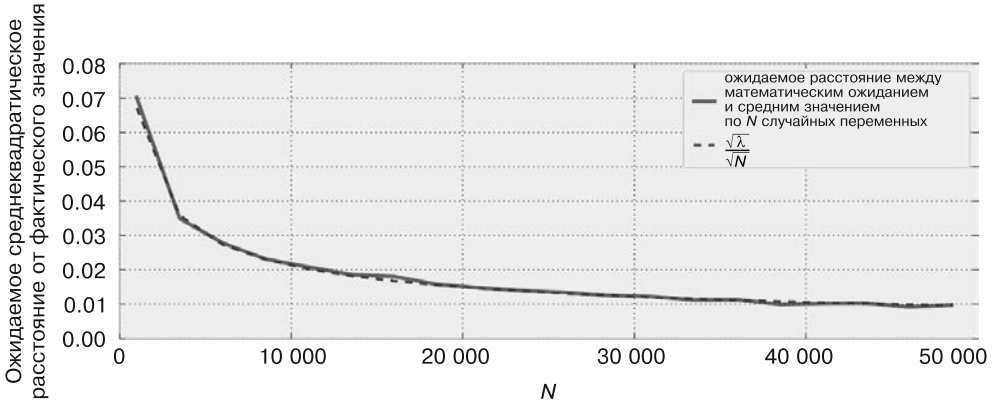

Рис. 4.2. Насколько быстро сходится среднее по выборкам?

Как и следовало предполагать, ожидаемое расстояние между выборочным средним и фактическим математическим ожиданием сокращается по мере роста N. Однако обратите внимание, что темпы сходимости снижаются, то есть для уменьшения расстояния с 0,020 до 0,015 (разница равна 0,005) было достаточно всего 10 000 дополнительных выборок, а для его уменьшения с 0,015 до 0,010 (разница тоже равна 0,005) необходимо уже 20 000 дополнительных выборок.

Оказывается, что эти темпы сходимости можно оценить. На рис. 4.2 есть график еще одной линии — функции $\frac{\sqrt{\lambda}}{\sqrt{N}}$. Этот выбор неслучаен. В большинстве ситуаций для заданной последовательности случайных переменных, распределенных аналогично Z, темп сходимости $E[Z]$ согласно закону больших чисел равен:

$$\frac{\sqrt{Var(Z)}}{\sqrt{N}}.$$

Знание удаленности (в среднем) от фактического значения для заданного большого N не помешает. С другой стороны, при байесовском анализе полезность этих знаний сомнительна: неопределенность для байесовского анализа не страшна, поэтому есть ли смысл со статистической точки зрения в нескольких дополнительных знаках после запятой? Получение выборок может потребовать столь малого количества вычислительных ресурсов, что увеличение N вполне допустимо.

4.2.3. Как вычислить Var(Z)

Дисперсия — это просто еще одно математическое ожидание, которое можно вычислить приближенно! После вычисления математического ожидания (посредством оценки в соответствии с законом больших чисел), которое мы обозначим μ, можно оценить и дисперсию:

$$\frac{1}{N}\sum_{i=1}^{N}(Z_i-\mu)^2 \to E[(Z-\mu)^2] = Var(Z).$$

4.2.4. Математические ожидания и вероятности

Связь между математическим ожиданием и оценкой вероятностей еще менее явная. Определим *индикаторную функцию* следующим образом:

$$\mathbb{1}_A(x) = \begin{cases} 1 & x \in A \\ 0 & else \end{cases}.$$

Далее согласно закону больших чисел при заданном достаточно большом количестве выборок X_i можно оценить вероятность события A, обозначаемую $P(A)$, по формуле:

$$\frac{1}{N}\sum_{i=1}^{N}\mathbb{1}_A(X_i) \to E[\mathbb{1}_A(X)] = P(A).$$

Опять же, эта формула совершенно понятна после короткого размышления. Индикаторная функция равна 1 только при возникновении события, так что суммируются лишь те случаи, когда событие имело место, а делится на общее число испытаний (сравните с тем, как обычно приближенно вычисляются вероятности с помощью частотного подхода). Например, пусть нам нужно оценить вероятность того, что $Z \sim \text{Exp}(0{,}5)$ больше 10 и у нас имеется множество выборок из распределения Exp(0,5).

$$P(Z>10) = \sum_{i=1}^{N}\mathbb{1}_{z>10}(Z_i).$$

```
import pymc as pm
N = 10000
print np.mean([pm.rexponential(0.5)>10 for i in range(N)])
```

```
[Output]:

0.0069
```

4.2.5. Какое отношение все это имеет к байесовской статистике

Точечные оценки (point estimates), о которых мы поговорим в главе 5, при байесовском выводе вычисляются на основе математических ожиданий. При байесовском выводе в более аналитической форме пришлось бы вычислять сложные математические ожидания, выраженные в виде многомерных интегралов. Больше не придется! При выборке непосредственно из апостериорного распределения нужно только вычислить средние значения, что значительно проще. Если же главное — точность, то оценить скорость сходимости можно по графикам, аналогичным приведенному на рис. 4.2. Так что, если требуется более высокая точность, нужно взять больше выборок из апостериорного распределения.

Когда можно остановиться и прекратить выборку из апостериорного распределения? Решение зависит от конкретных условий, в том числе от дисперсии выборок (напомню, что высокая дисперсия означает более медленную сходимость среднего значения).

Следует также понимать, когда закон больших чисел не работает. Как понятно из его названия, да и из внешнего вида наших графиков для малых N, закон больших чисел справедлив только для больших размеров выборки. Без этого асимптотические результаты доверия не заслуживают. Понимание того, в каких случаях закон больших чисел не работает, дает нам уверенность в том, какова должна быть наша степень *неуверенности*. Этот вопрос мы обсудим в следующем разделе.

4.3. Некорректная работа при малых числах

Закон больших чисел справедлив только при бесконечно больших N, на практике полностью недостижимых. Хотя этот закон — замечательный инструмент, было бы глупо применять его повсюду. Это иллюстрирует следующий пример.

4.3.1. Пример: агрегированные географические данные

Данные часто поступают на обработку в агрегированной форме. Например, они могут быть агрегированы по штатам, странам или городам. Конечно, демографические данные различаются для разных географических областей. Если данные представляют собой среднее значение какого-либо показателя по каждой из географических областей, нужно помнить о законе больших чисел и о том, что он *не всегда работает* для областей с маленьким населением.

Рассмотрим это на примере модельного набора данных. Пусть наш набор данных включает 5000 округов. Кроме того, все демографические показатели в каждом

из штатов равномерно распределены по отрезку от 100 до 1500 (то, откуда эти числа взялись, для наших целей неважно). Нам нужно оценить средний рост людей по округам. Хотя нам это неизвестно, но рост *не* меняется от округа к округу и распределение роста людей не зависит от округа, где они проживают.

$$\text{Рост} \sim \text{Normal}(150, 15).$$

Данные агрегируются на уровне округов, так что мы можем вычислить только среднее значение по округу. Как же будет выглядеть наш набор данных?

```
figsize(12.5, 4)
std_height = 15
mean_height = 150
n_counties = 5000
pop_generator = pm.rdiscrete_uniform
norm = pm.rnormal

# Генерируем какие-то фиктивные демографические данные
population = pop_generator(100, 1500, size = n_counties)

average_across_county = np.zeros(n_counties)
for i in range(n_counties):
    # Генерируем данные по людям и вычисляем среднее значение
    average_across_county[i] = norm(mean_height, 1./std_height**2,
                                    size=population[i]).mean()

# Находим округа с явно выделяющимися значениями роста жителей
i_min = np.argmin(average_across_county)
i_max = np.argmax(average_across_county)

# Строим график зависимости количества населения от среднего роста
plt.scatter(population, average_across_county, alpha=0.5, c="#7A68A6")
plt.scatter([population[i_min], population[i_max]],
            [average_across_county[i_min], average_across_county[i_max]],
            s=60, marker="o", facecolors="none",
            edgecolors="#A60628", linewidths=1.5,
            label=u"экстремальные значения роста")

plt.xlim(100, 1500)
plt.title(u"Зависимость среднего роста от количества населения")
plt.xlabel(u"Население округа")
plt.ylabel(u"Средний рост в округе")
plt.plot([100, 1500], [150, 150], color="k", label=u"фактический \
          ожидаемый рост", ls="--")
plt.legend(scatterpoints = 1);
```

Что видно из графика (рис. 4.3)? Без учета размера населения мы рискуем допустить колоссальную ошибку при выводе, а именно: если игнорировать размер населения, можно сказать, что округа с самыми низкими и высокими людьми были

правильно обведены кружками на рис. 4.3. Но этот вывод ложен по следующей причине. Рост в указанных двух округах вовсе *не* обязательно экстремальный, ведь вычисленное среднее значение при малом населении плохо отражает фактическое математическое ожидание населения (которое на самом деле должно быть равно $\mu = 150$). Размер выборки/размер населения/N — как его ни обзови — просто слишком мал, чтобы эффективно задействовать закон больших чисел.

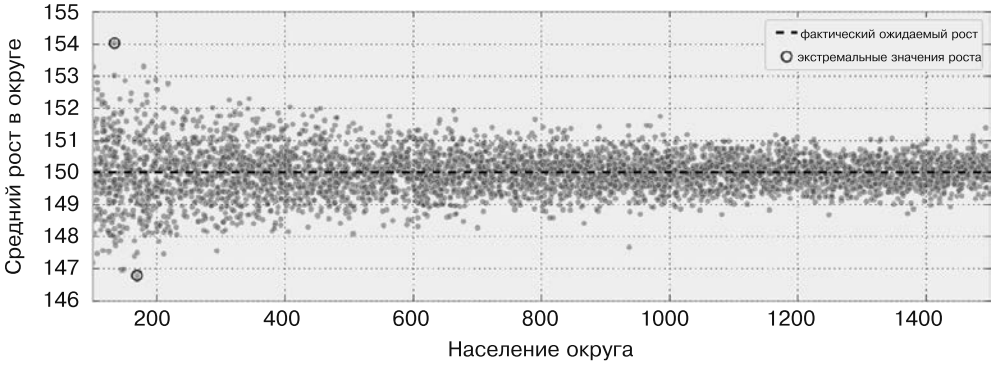

Рис. 4.3. Зависимость среднего роста от количества населения

Можно привести еще более весомые аргументы, свидетельствующие отнюдь не в пользу результатов этого вывода. Напомню, что показатели количества населения равномерно распределены по отрезку от 100 до 1500. Интуиция говорит нам, что округа с наиболее экстремальными показателями роста жителей также должны быть равномерно распределены по отрезку от 100 до 1500 и, разумеется, не должны зависеть от населения округа. Но это не так. Ниже приведено население округов с наиболее экстремальными показателями роста жителей.

```
print "Население 10 наименее 'населенных' округов: "
print population[np.argsort(average_across_county)[:10]]
print
print "Население 10 наиболее 'населенных' округов: "
print population[np.argsort(-average_across_county)[:10]]
```

```
[Output]:

Население 10 наименее 'населенных' округов:
[111 103 102 109 110 257 164 144 169 260]

Население 10 наиболее 'населенных' округов:
[252 107 162 141 141 256 144 112 210 342]
```

Показатели количества населения вовсе не распределены равномерно по отрезку от 100 до 1500. Закон больших чисел здесь совершенно не работает.

4.3.2. Пример: конкурс Kaggle (перепись населения США)

Ниже приведены данные переписи населения США 2010 года, когда население было разбито вплоть до уровня групп кварталов городов. Этот набор данных взят из конкурса по машинному обучению Kaggle, в котором участвовал я и несколько моих сослуживцев. Цель конкурса состояла в предсказании процента отправляемых обратно опросных листов переписи для групп кварталов, от 0 до 100, на основе переменных переписи (медианный доход, количество женщин в данной группе кварталов, количество трейлер-парков, среднее число детей и т. д.). На рис. 4.4 приведен график зависимости процента отправляемых обратно опросных листов переписи от населения группы кварталов:

```
figsize(12.5, 6.5)
data = np.genfromtxt("data/census_data.csv", skip_header=1,
                     delimiter=",")
plt.scatter(data[:,1], data[:,0], alpha=0.5, c="#7A68A6")
plt.title(u"Зависимость процента отправляемых листов переписи от населения")
plt.ylabel(u"Процент отправляемых обратно листов")
plt.xlabel(u"Население группы кварталов")
plt.xlim(-100, 15e3)
plt.ylim(-5, 105)

i_min = np.argmin(data[:,0])
i_max = np.argmax(data[:,0])

plt.scatter([data[i_min,1], data[i_max, 1]],
            [data[i_min,0], data[i_max,0]],
            s=60, marker="o", facecolors="none",
            edgecolors="#A60628", linewidths=1.5,
            label=u"точки экстремумов")

plt.legend(scatterpoints = 1);
```

Это типичное явление в статистике. Слово *«типичное»* относится к форме диаграммы рассеяния на рис. 4.4. Эта форма имеет вид треугольника, сужающегося по мере роста размера выборки (по мере того как усиливается справедливость закона больших чисел).

Возможно, я немного преувеличиваю и мне стоило назвать книгу *«С большими данными у вас не будет проблем!»*, но это опять пример проблемы, возникающей вследствие маленьких, а не больших наборов данных. Проще говоря, к маленьким наборам данных неприменим закон больших чисел, который можно спокойно использовать для больших наборов (то есть настоящих больших данных). Я уже упоминал ранее, что, как ни странно, задачи предсказания в сфере больших данных решаются с помощью относительно простых алгоритмов. Этот парадокс частично разрешается, если понимать, что закон больших чисел дает *устойчивые* решения —

такие, что увеличение или уменьшение выборки на несколько точек данных особенно на них не влияет. С другой стороны, добавление или исключение точек данных из маленького набора данных может привести к совершенно другим результатам.

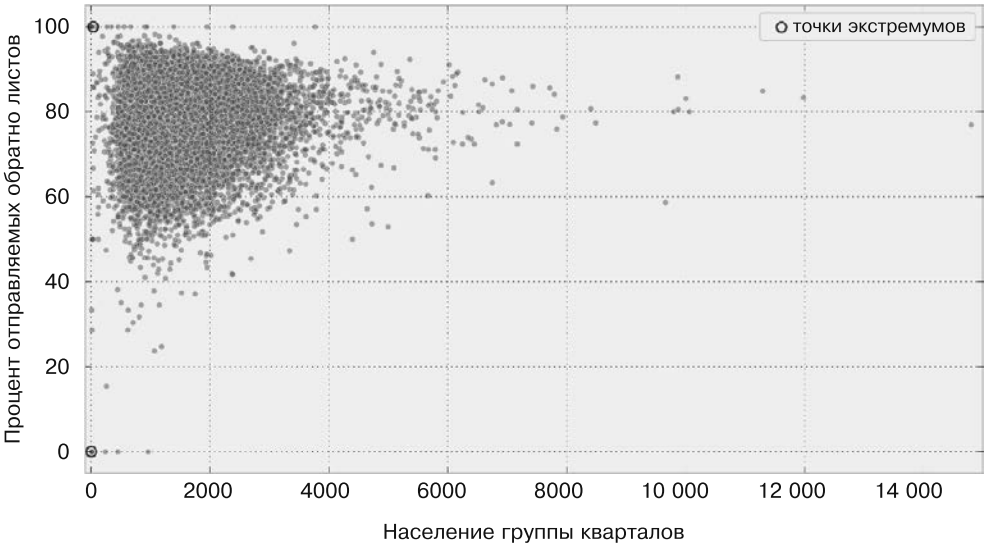

Рис. 4.4. Зависимость процента отправляемых обратно листов от переписи населения

Для дальнейшего получения информации о подводных камнях закона больших чисел я бы очень рекомендовал вам почитать прекрасную монографию «Самое опасное уравнение» (http://faculty.cord.edu/andersod/MostDangerousEquation.pdf).

4.3.3. Пример: сортировка комментариев на Reddit

Возможно, вы не согласны, что закон больших чисел применяется всеми, хотя и только неявно, при подсознательном принятии решений. Рассмотрим пример онлайн-рейтингов товаров. Часто ли вы доверяете среднему рейтингу в пять баллов на основе одного отзыва? Двух отзывов? Трех отзывов? Вы подсознательно понимаете, что при таком малом количестве отзывов средний рейтинг *плохо* отражает то, насколько товар на самом деле хорош/плох.

Вследствие этого возникают упущения при сортировке товаров и вообще при их сравнении. Для многих покупателей понятно, что сортировка результатов интерактивного поиска по рейтингу не слишком объективна, неважно, идет ли речь о книгах, видео или комментариях в Интернете. Зачастую находящиеся на первых местах фильмы или комментарии получают высокие оценки только за счет небольшого числа восторженных поклонников, а действительно хорошие фильмы

или комментарии скрыты на последующих страницах с якобы неидеальными рейтингами около 4,8. Что же с этим делать?

Рассмотрим популярный сайт Reddit (я осознанно не привожу на него ссылки, ведь Reddit печально знаменит тем, что затягивает пользователей, и я боюсь, что вы никогда не вернетесь к моей книге). На этом сайте есть множество ссылок на разные истории и картинки, причем комментарии к этим ссылкам тоже очень популярны. Пользователи сайта (которых обычно называют словом redditor[1]) могут голосовать за каждый комментарий или против него (так называемые *голоса «за»* (upvotes) и *голоса «против»* (downvotes)). Reddit по умолчанию сортирует комментарии по убыванию. Как же определить, какие комментарии лучшие? Обычно ориентируются на следующие несколько показателей.

1. *Популярность*. Комментарий считается хорошим, если за него подано много голосов. Проблемы при использовании этой модели начинаются в случае комментария с сотнями голосов «за» и тысячами «против». Хотя и очень популярный, этот комментарий, похоже, слишком неоднозначен, чтобы считаться «лучшим».
2. *Разница*. Можно воспользоваться *разницей* между количеством голосов «за» и «против». Это решает проблему, возникающую при использовании метрики «популярность», но не учитывает временну́ю природу комментариев. Комментарии могут быть отправлены через многие часы после публикации первоначальной ссылки. При этом возникает смещение, из-за которого самый высокий рейтинг получают вовсе не лучшие комментарии, а самые старые, успевшие накопить больше голосов «за», чем более новые.
3. *Поправка на время*. Рассмотрим метод, при котором разница голосов «за» и «против» делится на возраст комментария и получается *частота* (rate), например *величина разницы в секунду* или *в минуту*. Сразу же на ум приходит контрпример: при использовании варианта «в секунду» комментарий, оставленный секунду назад с одним голосом «за» окажется лучше, чем оставленный 100 секунд назад с 99 голосами «за». Избежать этой проблемы можно, если учитывать только комментарии, оставленные как минимум t секунд назад. Но как выбрать хорошее значение t? Значит ли это, что все комментарии, оставленные позднее, чем t секунд назад, плохи? Дело закончится сравнением неустойчивых величин с устойчивыми (новых и старых комментариев).
4. *Соотношение*. Ранжирование комментариев по соотношению числа голосов «за» к суммарно́му числу голосов «за» и «против». Такой подход устраняет проблему с временно́й природой комментариев, так что недавно оставленные комментарии с хорошими оценками получат высокий рейтинг с той же вероятностью, что и оставленные давно, при условии, что у них относительно высокое отношение голосов «за» к общему числу голосов. Проблема с этим методом состоит в том,

[1] Игра слов со словом editor — «редактор».

что комментарий с одним голосом «за» (отношение = 1,0) окажется лучше, чем комментарий с 999 голосами «за» и одним «против» (отношение = 0,999), хотя очевидно, что второй из этих комментариев *скорее* окажется лучшим.

Я неспроста написал *«скорее»*. Может оказаться и так, что первый комментарий с одним-единственным голосом «за» действительно лучше, чем второй, с 999 голосами «за». С этим утверждением трудно согласиться, ведь мы не знаем, какими могли бы быть 999 потенциальных следующих голосов для первого комментария. Скажем, он мог получить в результате еще 999 голосов «за» и ни одного голоса «против» и оказаться лучше, чем второй, хотя такой сценарий и не слишком вероятен.

На самом деле нам нужно оценить *фактическое соотношение голосов «за»*. Отмечу, что это вовсе не то же, что наблюдаемое соотношение голосов «за»; фактическое соотношение голосов «за» скрыто, наблюдаем мы только число голосов «за» по сравнению с голосами «против» (фактическое соотношение голосов «за» можно рассматривать как вероятность получения данным комментарием голоса «за», а не «против»). Благодаря закону больших чисел можно с уверенностью утверждать, что у комментария с 999 голосами «за» и одним «против» фактическое соотношение голосов «за», вероятно, будет близко к 1. С другой стороны, мы гораздо менее уверены в том, каким окажется фактическое соотношение голосов «за» для комментария с одним голосом «за». Похоже, что это байесовская задача.

Один из способов определить априорное распределение соотношения голосов «за» — изучить историю распределения соотношений голосов «за». Это можно сделать с помощью скрапинга комментариев Reddit и последующего определения распределения. Впрочем, у этого метода есть несколько недостатков.

1. *Асимметричные данные*. Число голосов у абсолютного большинства комментариев очень невелико, вследствие чего соотношения у многих комментариев будут близки к экстремальным (см. «треугольный» график в примере с набором данных Kaggle на рис. 4.4) и распределение окажется сильно «перекошено». Можно попробовать учитывать лишь комментарии, число голосов у которых превышает определенное пороговое значение. Но и здесь возникают сложности. Приходится искать баланс между числом доступных комментариев, с одной стороны, и более высоким пороговым значением с соответствующей точностью соотношения — с другой.
2. *Смещенные (содержащие систематическую погрешность) данные*. Reddit состоит из множества подфорумов (subreddits). Два примера: *r/aww* с фотографиями забавных животных и *r/politics*. Более чем вероятно, что поведение пользователей при комментировании этих двух подфорумов Reddit будет кардинально различаться: в первом из них посетители, скорее всего, будут умиляться и вести себя дружелюбно, что приведет к большему числу голосов «за», по сравнению со вторым, где мнения в комментариях, вероятно, будут расходиться.

В свете вышеизложенного мне кажется, что имеет смысл воспользоваться равномерным априорным распределением.

Теперь мы можем вычислить апостериорное распределение фактического соотношения голосов «за». Сценарий `comments_for_top_reddit_pic.py` служит для скрапинга комментариев из текущей наиболее популярной картинки Reddit. В следующем коде мы произвели скрапинг комментариев Reddit, относящихся к картинке [3]: http://i.imgur.com/OYsHKlH.jpg.

```
from IPython.core.display import Image
# С помощью добавления числа к вызову %run
# можно получить i-ю по рейтингу фотографию.
%run top_pic_comments.py 2
```

```
[Output]:

Title of submission:
Frozen mining truck
http://i.imgur.com/OYsHKlH.jpg
```

```
"""
Contents: массив текстов всех комментариев к картинке Votes: двумерный
массив NumPy голосов "за" и "против" для каждого комментария
"""
n_comments = len(contents)
comments = np.random.randint(n_comments, size=4)
print "Несколько комментариев (из общего числа в %d) \n
      -----------"%n_comments
for i in comments:
    print '"' + contents[i] + '"'
    print "голоса "за"/"против": ",votes[i,:]
    print
```

```
[Output]:

Несколько комментариев (из общего числа 77)
-----------
"Do these trucks remind anyone else of Sly Cooper?"
голоса "за"/"против": [2 0]

"Dammit Elsa I told you not to drink and drive."
голоса "за"/"против": [7 0]

"I've seen this picture before in a Duratray (the dump box supplier) brochure..."
голоса "за"/"против": [2 0]

"Actually it does not look frozen just covered in a layer of wind packed snow."
голоса "за"/"против": [120 18]
```

При N голосах и заданном фактическом соотношении голосов «за» p число голосов «за» напоминает биномиальную случайную переменную с параметрами p и N

(дело в том, что фактическое соотношение голосов «за» эквивалентно вероятности подачи голоса «за» по сравнению с голосом «против» при N возможных голосах/испытаниях). Создадим функцию для байесовского вывода по p в отношении набора голосов «за»/«против» конкретного комментария.

```
import pymc as pm

def posterior_upvote_ratio(upvotes, downvotes, samples=20000):
    """
    Эта функция принимает в качестве параметров количество
    голосов "за" и "против", полученных конкретным комментарием,
    а также количество выборок, которое нужно вернуть пользователю.
    Предполагается, что априорное распределение равномерно.
    """
    N = upvotes + downvotes
    upvote_ratio = pm.Uniform("upvote_ratio", 0, 1)
    observations = pm.Binomial("obs", N, upvote_ratio,
                               value=upvotes, observed=True)
    # Обучение; сначала выполняем метод MAP, поскольку он не требует
    # больших вычислительных затрат и приносит определенную пользу.
    map_ = pm.MAP([upvote_ratio, observations]).fit()
    mcmc = pm.MCMC([upvote_ratio, observations])
    mcmc.sample(samples, samples/4)
    return mcmc.trace("upvote_ratio")[:]
```

Далее приведены получившиеся в результате апостериорные распределения.

```
figsize(11., 8)
posteriors = []
colors = ["#348ABD", "#A60628", "#7A68A6", "#467821", "#CF4457"]
for i in range(len(comments)):
    j = comments[i]
    label = u'(%d за:%d против)\n%s...'%(votes[j, 0], votes[j,1],
                                         contents[j][:50])
    posteriors.append(posterior_upvote_ratio(votes[j, 0], votes[j,1]))
    plt.hist(posteriors[i], bins=18, normed=True, alpha=.9,
             histtype="step", color=colors[i%5], lw=3, label=label)
    plt.hist(posteriors[i], bins=18, normed=True, alpha=.2,
             histtype="stepfilled", color=colors[i], lw=3)

plt.legend(loc="upper left")
plt.xlim(0, 1)
plt.ylabel(u"Плотность")
plt.xlabel(u"Вероятность голоса 'за'")
plt.title(u"Апостериорные распределения соотношений голосов 'за' \
          для различных комментариев");
```

```
[Output]:

[***************100%*****************] 20000 of 20000 complete
```

Как видно из рис. 4.5, некоторые распределения сильно «сжаты», у других же — сравнительно длинные «хвосты», выражающие то, что мы точно не знаем, чему равно фактическое соотношение голосов «за».

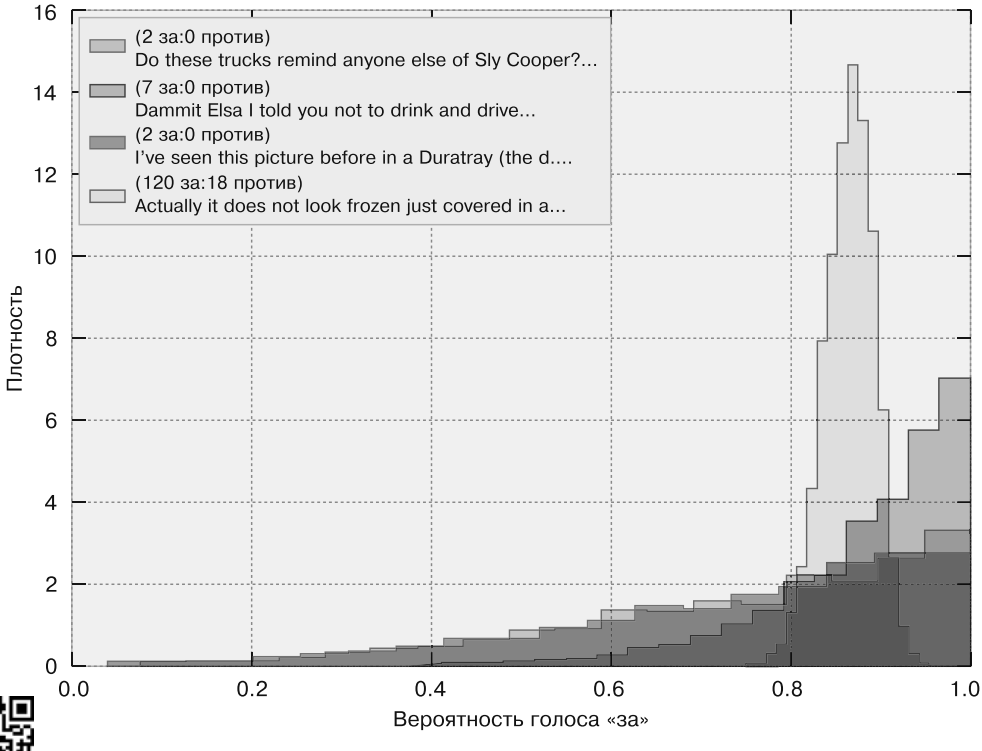

Рис. 4.5. Апостериорные распределения соотношений голосов «за» для различных комментариев

4.3.4. Сортировка

До сих пор мы игнорировали основную цель нашего примера: сортировку комментариев от лучшего к худшему. Конечно, невозможно сортировать распределения; сортировать нужно скалярные значения. Предусмотрено множество способов извлечь сущность распределения в виде скаляра; например, можно выразить суть распределения через его математическое ожидание, или среднее значение. Впрочем, среднее значение для этого подходит плохо, поскольку этот показатель не учитывает неопределенность распределений.

Я рекомендовал бы воспользоваться **95%-ным наименее правдоподобным значением** (least plausible value), которое определяется как значение с лишь 5%-ной вероятностью того, что фактическое значение параметра ниже его (ср. с нижней границей байесовского доверительного интервала). Далее мы строим графики

апостериорных распределений с указанным 95%-ным наименее правдоподобным значением (рис. 4.6).

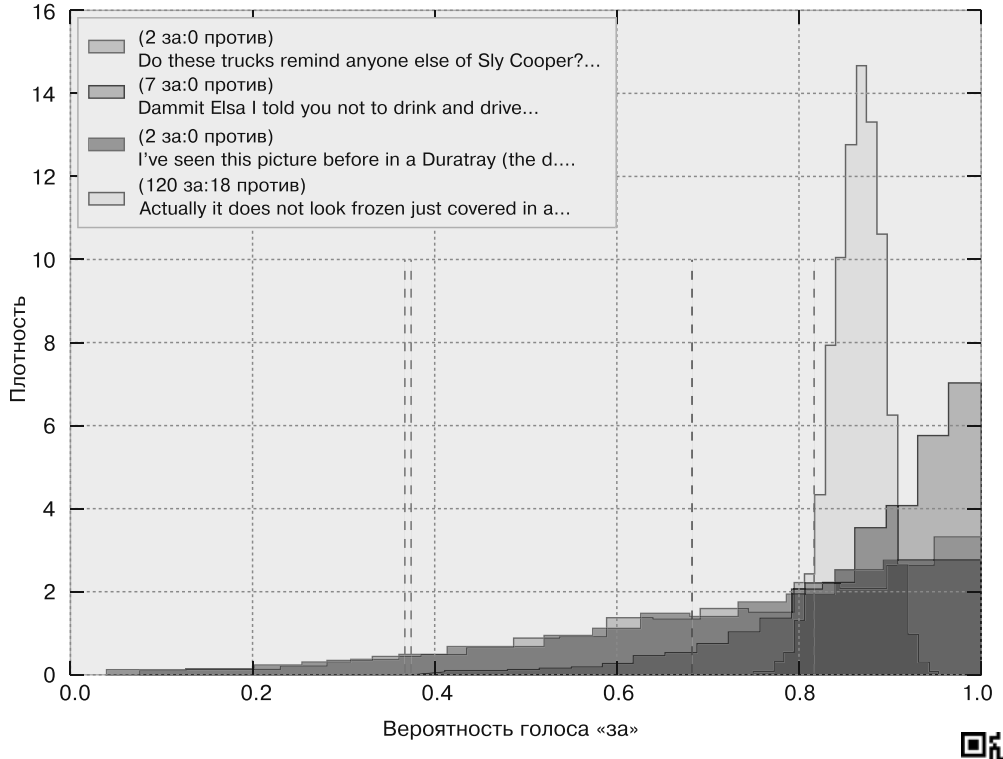

Рис. 4.6. Апостериорные распределения соотношений голосов «за» для различных комментариев

```
N = posteriors[0].shape[0]
lower_limits = []
for i in range(len(comments)):
    j = comments[i]
    label = '(%d за:%d против)\n%s…'%(votes[j, 0], votes[j,1],
                                      contents[j][:50])
    plt.hist(posteriors[i], bins=20, normed=True, alpha=.9,
             histtype="step", color=colors[i], lw=3, label=label)
    plt.hist(posteriors[i], bins=20, normed=True, alpha=.2,
             histtype="stepfilled", color=colors[i], lw=3)
    v = np.sort(posteriors[i])[int(0.05*N)]
    plt.vlines(v, 0, 10 , color=colors[i], linestyles="—",
               linewidths=3)
    lower_limits.append(v)

plt.legend(loc="upper left")
plt.ylabel(u"Плотность")
plt.xlabel(u"Вероятность голоса 'за'")
```

```
plt.title(u"Апостериорные распределения соотношений голосов 'за' \
          для различных комментариев");

order = np.argsort(-np.array(lower_limits))
print order, lower_limits
```

```
[Output]:

[3 1 2 0] [0.36980613417267094, 0.68407203257290061,
    0.37551825562169117, 0.8177566237850703]
```

Лучшими, согласно нашей процедуре, окажутся те комментарии, для которых наиболее высока вероятность получения высокого процента голосов «за». Визуально это комментарии с ближайшим к единице 95%-ным наименее правдоподобным значением. На рис. 4.6 95%-ное наименее правдоподобное значение изображено с помощью вертикальных линий.

Почему же сортировка на основе этого показателя — такая хорошая идея? Упорядочение в соответствии с 95%-ным наименее правдоподобным значением означает максимальную осторожность в объявлении комментариев лучшими. То есть даже при наихудшем сценарии, если мы сильно переоценили соотношение голосов «за», гарантируется, что лучшие комментарии окажутся сверху. При таком упорядочении обеспечиваются следующие весьма естественные свойства.

1. Из двух комментариев с одинаковым наблюдаемым соотношением голосов «за» лучшим будет признан комментарий с бóльшим числом голосов (поскольку выше уверенность в более высоком соотношении для него).
2. Из двух комментариев с одинаковым количеством голосов лучшим считается комментарий с бóльшим числом голосов «за».

4.3.5. Но для режима реального времени это слишком медленно!

Согласен. Вычисление апостериорного распределения для каждого из комментариев занимает слишком много времени, и ко времени его окончания данные, вероятно, уже поменяются. Математические нюансы я отложу до приложения к данной главе, но для быстрого вычисления нижней границы рекомендую воспользоваться следующей формулой:

$$\frac{a}{a+b} - 1,65\sqrt{\frac{ab}{(a+b)^2(a+b+1)}},$$

где:

$$a = 1 + u;$$
$$b = 1 + d.$$

Здесь u — число голосов «за», а d — число голосов «против». Эта формула представляет собой сокращенный способ байесовского вывода, о котором я расскажу в главе 6, когда мы будем подробнее говорить об априорных распределениях.

```
def intervals(u,d):
    a = 1. + u
    b = 1. + d
    mu = a/(a+b)
    std_err = 1.65*np.sqrt((a*b)/((a+b)**2*(a+b+1.)))
    return (mu, std_err)

print "Приближенные нижние границы:"
posterior_mean, std_err = intervals(votes[:,0],votes[:,1])
lb = posterior_mean - std_err
print lb
print
print "Список 40 лучших комментариев, отсортированных по приближенным нижним
границам:"
print
order = np.argsort(-lb)
ordered_contents = []
for i in order[:40]:
    ordered_contents.append(contents[i])
    print votes[i,0], votes[i,1], contents[i]
    print "-------------"
```

```
[Output]:

Приближенные нижние границы:
[ 0.83167764 0.8041293  0.8166957  0.77375237 0.72491057 0.71705212
  0.72440529 0.73158407 0.67107394 0.6931046  0.66235556 0.6530083
  0.70806405 0.60091591 0.60091591 0.66278557 0.60091591 0.60091591
  0.53055613 0.53055613 0.53055613 0.53055613 0.53055613 0.43047887
  0.43047887 0.43047887 0.43047887 0.43047887 0.43047887 0.43047887
  0.43047887 0.43047887 0.43047887 0.43047887 0.43047887 0.43047887
  0.43047887 0.43047887 0.43047887 0.47201974 0.45074913 0.35873239
  0.3726793  0.42069919 0.33529412 0.27775794 0.27775794 0.27775794
  0.27775794 0.27775794 0.27775794 0.27775794 0.13104878 0.13104878 0.27775794
  0.27775794 0.27775794 0.27775794 0.27775794 0.27775794 0.27775794
  0.27775794 0.27775794 0.27775794 0.27775794 0.27775794 0.27775794
  0.27775794 0.27775794 0.27775794 0.27775794 0.27775794 0.27775794
  0.27775794 0.27775794 0.27775794 0.27775794 0.27775794]

Список 40 лучших комментариев, отсортированных по приближенным нижним границам:

327 52 Can you imagine having to start that? I've fired up much smaller
    equipment when its around 0° out and its still a pain. It would
    probably take a crew of guys hours to get that going. Do they have
    built in heaters to make it easier? You'd think they would just let
    them idle overnight if they planned on running it the next day
    though.
```

```
-------------
120 18 Actually it does not look frozen just covered in a layer of wind
    packed snow.
-------------
70 10 That's actually just the skin of a mining truck. They shed it
    periodically like snakes do.
-------------
76 14 The model just hasn't been textured yet!
-------------
21 3 No worries, [this](http://imgur.com/KeSYJud) will help.
-------------
7 0 Dammit Elsa I told you not to drink and drive.
-------------
88 23 Speaking of mining...[BAGGER 288!](http://www.youtube.com/
    watch?v=azEvfD4C6ow)
-------------
112 32 Wonder why OP has 31,944 link karma but so few submissions?
    /u/zkool may have the worst case of karma addiction I'm aware of.

title | points | age | /r/ | comnts
:--|:--|:--|:--|:--
[Frozen mining truck](http://www.reddit.com/r/pics/comments/1mrqvh/
    frozen mining truck/) | 2507 | 4^mos | pics | 164
[Frozen mining truck](http://www.reddit.com/r/pics/comments/1cutbw/
    frozen mining truck/) | 16 | 9^mos | pics | 4
[Frozen mining truck](http://www.reddit.com/r/pics/comments/vvcrv/
    frozen mining truck/) | 439 | 1^yr | pics | 21
[Meanwhile, in New Zealand...](http://www.reddit.com/r/pics/comments/
    ir1pl/meanwhile in new zealand/) | 39 | 2^yrs | pics | 12
[Blizzardy day](http://www.reddit.com/r/pics/comments/1uiu3y/
    blizzardy day/) | 7 | 19^dys | pics | 3

*[Source: karmadecay](http://karmadecay.com/r/pics/comments/1w454i/
    frozen mining truck/)*
-------------
11 1 This is what it's typically like, living in Alberta.
-------------
6 0 That'd be a haul truck. Looks like a CAT 793. We run em at the site
    I work at, 240ton carrying capacity.
-------------
22 5 Taken in Fort Mcmurray Ab!
-------------
9 1 "EXCLUSIVE: First look at "Hoth" from the upcoming 'Star Wars:
    Episode VII'"
-------------
32 9 This is the most fun thing to drive in GTA V.
-------------
5 0 it reminds me of the movie "moon" with sam rockwell.
-------------
```

```
4 0 Also frozen drill rig.
-------------
4 0 There's just something awesome about a land vehicle so huge that
    it warrants a set of stairs on the front of it. I find myself
    wishing I were licensed to drive it.
-------------
4 0 Heaters all over the components needing heat:
    http://www.arctic-fox.com/fuel-fluid-warming-products/
    diesel-fired-coolant-pre-heaters
-------------
4 0 Or it is just an amazing snow sculpture!
-------------
3 0 I have to tell people about these awful conditions... Too bad I'm
    Snowden.
-------------
3 0 Someone let it go
-------------
3 0 Elsa, you can't do that to people's trucks.
-------------
3 0 woo Alberta represent
-------------
3 0 Just thaw it with love
-------------
6 2 Looks like the drill next to it is an IR DM30 or DM45. Good rigs.
-------------
4 1 That's the best snow sculpture I've ever seen.
-------------
2 0 [These](http://i.imgur.com/xYuwk5I.jpg) are used for removing
    the ice.
-------------
2 0 Please someone post frozen Bagger 288
-------------
2 0 It's kind of cool there are trucks so big they need both a
    ladder and a staircase...
-------------
2 0 Eight miners are just out of frame hiding inside a tauntaun.
-------------
2 0 http://imgur.com/gallery/Fxv3Oh7
-------------
2 0 It would take a god damn week just to warm that thing up.
-------------
2 0 Maybe /r/Bitcoin can use some of their mining equipment to heat
    this guy up!
-------------
2 0 Checkmate Jackie Chan
-------------
2 0 I've seen this picture before in a Duratray (the dump box supplier)
    brochure...
-------------
```

```
2 0 The Texas snow has really hit hard!
-------------
2 0 I'm going to take a wild guess and say the diesel is gelled.
-------------
2 0 Do these trucks remind anyone else of Sly Cooper?
-------------
2 0 cool
-------------
```

Чтобы наглядно продемонстрировать эту упорядоченность, мы построим графики апостериорных средних значений и границ с сортировкой по нижней границе. Обратите внимание, что на рис. 4.7 планки погрешности отсортированы по левой границе (как мы говорили выше, это оптимальный способ определения упорядоченности), так что никакой закономерности в отмеченных точками средних значениях не наблюдается.

```
r_order = order[::-1][-40:]
plt.errorbar(posterior_mean[r_order], np.arange(len(r_order)),
             xerr=std_err[r_order],xuplims=True, capsize=0, fmt="o",
             color="#7A68A6")
plt.xlim(0.3, 1)
plt.yticks(np.arange(len(r_order)-1,-1,-1),
           map(lambda x: x[:30].replace("\n",""), ordered_contents));
```

Из рис. 4.7 видно, почему не имеет смысла сортировать по средним значениям.

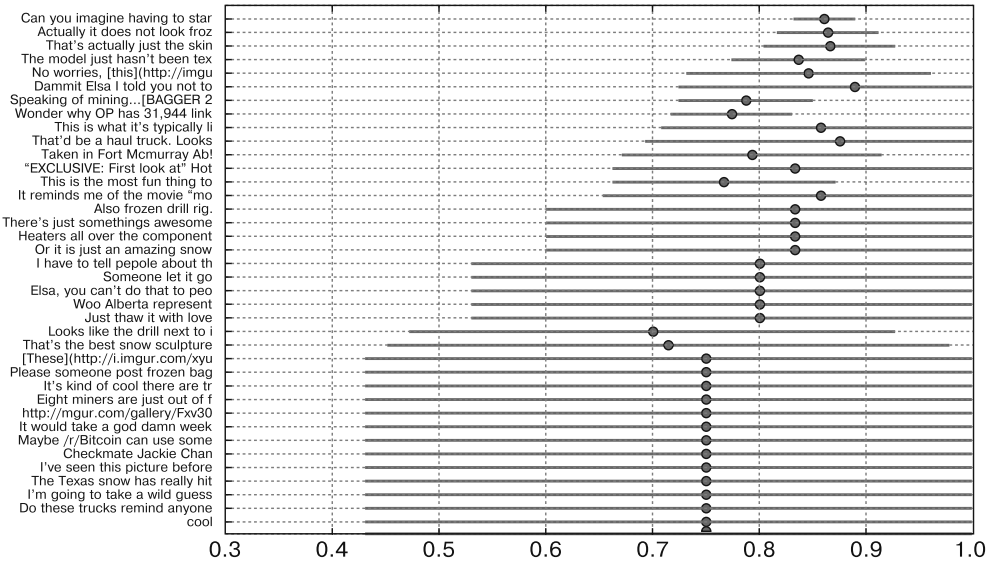

Рис. 4.7. Сортировка самых рейтинговых комментариев по нижней границе

4.3.6. Расширение на системы оценки с присвоением звезд

Вышеописанная процедура отлично работает для схем с голосами «за»/«против», но что делать в случае систем с рейтингом на основе звезд, например пятизвездочных систем рейтинга? Те же проблемы возникают, если использовать просто среднее значение: сущность с двумя отличными оценками окажется лучше, чем сущность с несколькими тысячами отличных оценок и одной более низкой оценкой.

Задачу с голосами «за»/«против» можно считать бинарной: 0 соответствует голосу «против», а 1 — голосу «за». Систему рейтинга на основе N звезд можно рассматривать как более непрерывную версию этой задачи, где оценка n звезд эквивалентна оценке n / N. Например, в пятизвездочной системе оценка две звезды соответствует 0,4. Идеальный рейтинг равен 1. Можно воспользоваться вышеприведенной формулой, только с другими определениями a и b.

$$\frac{a}{a+b} - 1,65\sqrt{\frac{ab}{(a+b)^2(a+b+1)}},$$

где:

$$a = 1 + S;$$
$$b = 1 + N - S.$$

Здесь N — число указавших рейтинг пользователей, а S — сумма всех рейтингов при вышеупомянутой схеме эквивалентности.

4.4. Выводы

Хотя закон больших чисел — замечательная вещь, он справедлив (в полном соответствии со своим названием) лишь для больших выборок. Как вы видели, на вывод существенно влияет учет или не учет *формы данных*.

1. Гарантировать применимость закона больших чисел при аппроксимации математических ожиданий (которая нам предстоит в главе 5) можно путем извлечения (без особых вычислительных затрат) большого числа выборок из апостериорных распределений.
2. Байесовский вывод учитывает, что при маленьком размере выборки результаты могут оказаться совершенно произвольными. В этом случае апостериорное распределение будет растянутым, а не сжатым. Следовательно, нужно обеспечить возможность исправления вывода.

3. Если не учесть размер выборки и попытаться отсортировать неустойчивые объекты, можно получить аномальную упорядоченность. Приведенный в подразделе 4.3.3 метод позволяет решить эту проблему.

4.5. Приложение

4.5.1. Дифференцирование формулы сортировки комментариев

По существу, мы используем априорное бета-распределение (с параметрами $a = 1$, $b = 1$, то есть равномерное распределение) и биномиальную функцию правдоподобия с наблюдениями u, $N = u + d$. Значит, наше апостериорное распределение будет бета-распределением с параметрами $a' = 1 + u$, $b' = 1 + (N - u) = 1 + d$. Далее необходимо найти значение x, такое, что вероятность 0,05 меньше, чем x. Обычно для этого производится обращение интегральной функции распределения (cumulative distribution function, CDF), но, хотя CDF бета-распределения при целочисленных параметрах и известно, оно требует суммирования [1].

Вместо этого мы аппроксимируем бета-распределение нормальным. Среднее значение бета-распределения равно $\mu = a' / (a' + b')$, а дисперсия равна:

$$\sigma^2 = \frac{a'b'}{(a'+b')^2(a'+b'+1)}.$$

Следовательно, для получения приближенной нижней границы необходимо решить такое уравнение для x:

$$0,05 = \Phi\left(\frac{(x-\mu)}{\sigma}\right),$$

где Φ — интегральная функция распределения для нормального распределения.

4.6. Упражнения

1. Как бы вы оценили показатель $E[\cos X]$, где $X \sim \text{Exp}(4)$? А $E[\cos X | X < 1]$ (то есть математическое ожидание при условии, что X меньше 1)?
2. Следующая таблица была приведена в статье «Вперед, за трехочковым: предсказание вероятности удачного филд-гола с помощью логистической регрессии» [2]. В ней кикеры (игроки, бьющие по мячу) в американском футболе расположены в соответствии с процентом попаданий в ворота. В чем исследователи ошиблись в этой таблице?

Позиция	Кикер	% попаданий	Количество ударов
1	Garrett Hartley	87,7	57
2	Matt Stover	86,8	335
3	Robbie Gould	86,2	224
4	Rob Bironas	86,1	223
5	Shayne Graham	85,4	254
...	...	...	...
51	Dave Rayner	72,2	90
52	Nick Novak	71,9	64
53	Tim Seder	71,0	62
54	Jose Cortez	70,7	75
55	Wade Richey	66,1	56

В августе 2013 года было опубликовано сообщение в блоге (https://bpodgursky.com/2013/08/21/average-income-per-programming-language/) [4] относительно тенденций среднего дохода программистов, пишущих на различных языках программирования. Ниже приведена итоговая таблица. Видите что-нибудь необычное, касающееся экстремальных значений?

Язык программирования	Средний доход семьи, доллары	Данные наблюдений
Puppet	87 589,29	112
Haskell	89 973,82	191
PHP	94 031,19	978
CoffeeScript	94 890,80	435
VimL	94 967,11	532
Shell	96 930,54	979
...	...	...
Scala	101 460,91	243
ColdFusion	101 536,70	109
Objective-C	101 801,60	562
Groovy	102 650,86	116
Java	103 179,39	1402
XSLT	106 199,19	123
ActionScript	108 119,47	113

4.6.1. Ответы

1.
```
import scipy.stats as stats

exp = stats.expon(scale=4)
N = 1e5
X = exp.rvs(N)

# E[cos(X)]
print (cos(X)).mean()
# E[cos(X) | X<1]
print (cos(X[X<1])).mean()
```

2. В обеих таблицах вычисленные сводные показатели сортируются «наивно» (конверсии в первой и средние значения во второй), без учета размера используемой для их вычисления выборки. Из-за этого возникают проблемы; например, в таблице с кикерами Garrett Hartley явно не лучший. Лучший — Matt Stover. В таблице с зарплатами оба экстремальных значения относятся к языкам программирования с малым числом наблюдений. Наивно (и ошибочно) было бы делать из этой сводной таблицы вывод о том, что компаниям приходится платить больше для привлечения программистов на редких языках, поскольку количество пишущих на них людей меньше.

4.7. Библиография

1. Бета-распределение // Википедия, Свободная энциклопедия [Электронный ресурс]. — Режим доступа: https://ru.wikipedia.org/wiki/Бета-распределение.
2. *Clark T. K., Johnson A. W., Stimpson A. J.* Going for Three: Predicting the Likelihood of Field Goal Success with Logistic Regression. Presented at the 7th Annual MIT Sloan Sports Analytics Conference, Cambridge, MA, March 1–2, 2013. Cambridge, MA: MIT Sloan Sports Analytics Conference, http://www.sloansportsconference.com/wp-content/uploads/2013/Going%20for%20Three%20Predicting%20the%20Likelihood%20of%20Field%20Goal%20Success%20with%20Logistic%20Regression.pdf.
3. Commentary surrounding Imgur image posted by user Zcool. Frozen Mining Truck. Reddit. http://www.reddit.com/r/pics/comments/1w454i/frozen_mining_truck/.
4. *Podgursky B.* Average Income per Programming Language. http://bpodgursky.com/2013/08/21/average-income-per-programming-language/ и https://bpodgursky.com/2013/08/22/updates-to-language-vs-income-breakdown-post/.

5 Что лучше: потерять руку или ногу?

5.1. Введение

Специалисты по статистике — типичные пессимисты. Вместо того чтобы подсчитывать выигрыши, они оценивают потери. На самом деле они даже рассматривают выигрыши как *отрицательные потери*. Но самое интересное, *как* они оценивают потери.

Например, рассмотрим следующую ситуацию.

> Метеорологу нужно предсказать вероятность того, что на его родной город обрушится ураган. Согласно полученной им оценке с уровнем доверия 95 % вероятность того, что ураган *не* заденет его город, составляет от 99 до 100 %. Он очень доволен достигнутой точностью и говорит городским властям, что полная эвакуация не потребуется. К сожалению, ураган все же обрушивается на город, и тот оказывается затоплен.

Этот условный пример демонстрирует недостаток применения чистой метрики точности для оценки результатов. При использовании метрики, делающей акцент на точности оценивания, каким бы логичным и объективным решением это ни казалось, упускается из виду главное, ради чего, собственно, и производится статистический вывод: результаты вывода. Более того, нам нужен метод, который бы учитывал размеры выигрышей от принятых решений, а не только точность оценки. Короче говоря: «Лучше оценка приблизительная, но правильная, чем точная, но неправильная в принципе» [1].

5.2. Функции потерь

Давайте познакомимся с тем, что специалисты, занимающиеся статистикой и теорией принятия решений, называют *функциями потерь* (loss functions). Функция потерь — функция от фактического значения параметра, оценка которого имеет вид:

$$L(\theta,\hat{\theta}) = f(\theta,\hat{\theta}).$$

Важно отметить, что функции потерь служат мерой того, насколько плоха текущая оценка: чем больше потери, тем хуже оценка относительно этой функции потерь. Простейший и часто встречающийся пример функции потерь: *функция потерь на основе среднеквадратичной ошибки* (squared-error loss). Она растет пропорционально квадрату разницы и используется в таких методах оценки, как линейная регрессия, вычисления несмещенных статистических показателей, и многих областях машинного обучения:

$$L(\theta, \hat{\theta}) = (\theta - \hat{\theta})^2.$$

Мы также будем использовать асимметричную функцию потерь на основе квадратичной ошибки:

$$L(\theta, \hat{\theta}) = \begin{cases} (\theta - \hat{\theta})^2, & \hat{\theta} < \theta \\ c(\theta - \hat{\theta})^2, & \hat{\theta} \geq \theta, \quad 0 < c < 1. \end{cases}$$

Она отражает тот факт, что лучше получить оценку большую, чем фактическое значение, чем меньшую. Полезной такая функция может оказаться, например, при оценке веб-трафика на следующий месяц, когда лучше завысить значение во избежание недостаточного выделения серверных ресурсов.

Недостаток функции потерь на основе квадратичной ошибки состоит в том, что сильно отклоняющимся аномальным значениям придается непропорционально большой вес. Дело в том, что по мере отклонения оценки потери растут квадратично, а не линейно. То есть штраф при удаленности на три единицы будет гораздо меньше, чем на пять, но не намного больше, чем на одну, хотя разница одинакова:

$$\frac{1^2}{3^2} < \frac{3^2}{5^2}, \text{ хотя } 3 - 1 = 5 - 3.$$

Такая функция потерь означает, что большая погрешность — очень плохо. Более робастной является *функция потерь на основе абсолютного значения ошибки*, растущая линейно с ростом ошибки. Она часто используется в машинном обучении и робастной статистике:

$$L(\theta, \hat{\theta}) = \left|\theta - \hat{\theta}\right|.$$

В числе других часто используемых функций потерь следующие: $L(\theta, \hat{\theta}) = 1_{\hat{\theta} \neq \theta}$ — часто применяемая в алгоритмах классификации машинного обучения *бинарная функция потерь* (zero-one loss); $L(\theta, \hat{\theta}) = -\hat{\theta}\log(\theta) - (1 - \hat{\theta})$, $\hat{\theta} \in 0, 1$, $\theta \in [0, 1]$, называемая *логарифмической функцией потерь* (log-loss) и также используемая в машинном обучении.

Исторически стимулами к использованию функций потерь были математическая простота (1) и робастность (2) (в том смысле, что функция потерь — объектив-

ная мера потерь). Первый стимул на самом деле не давал задействовать функции потерь во всей полноте. Благодаря тому что компьютерам неважно математическое удобство/неудобство, появляется возможность создания пользовательских функций потерь, чем мы воспользуемся в полной мере далее в этой главе.

Что касается второго стимула, вышеприведенные функции потерь действительно являются объективными, представляя собой в большинстве случаев функцию от разницы оценки и фактического значения параметра, причем не зависящую от знака этой разницы или от размера выигрыша вследствие выбора данной оценки. Однако последний нюанс — независимость от размера выигрыша — приводит к весьма странным результатам. Вернемся к примеру с ураганом. Предсказание нашего специалиста-статистика эквивалентно тому, что вероятность урагана находится между 0 и 1 %. Но если бы он вместо точности обратил внимание на возможные исходы (99 % — вероятность того, что затопления не будет, 1% — вероятность того, что будет), то мог бы дать властям другие рекомендации.

Если обратить больше внимания не на точность оценки параметров, а на получаемые в результате этой оценки исходы, можно будет оптимизировать оценки специально под нашу задачу. Для этого нам нужно создать новые функции потерь, отражающие наши цели и возможные исходы. Вот несколько примеров более многообещающих функций потерь.

- Функция потерь $L(\theta,\hat{\theta}) = \frac{\left|\theta-\hat{\theta}\right|}{\theta(1-\theta)}$, $\hat{\theta}$, $\theta \in [0, 1]$ придает особое значение близости оценки к 0 или 1, ведь если фактическое значение θ близко к 0 или 1, то потери будут очень велики, разве что $\hat{\theta}$ тоже близко к 0 или 1. Такую функцию потерь может применять в своей работе политический эксперт, от которого ждут уверенных ответов «да»/«нет». Эта функция потерь отражает необходимость для такого эксперта при близости фактического значения параметра к 1 (например, если определенный политический исход весьма вероятен) продемонстрировать уверенность в своих словах, чтобы не прослыть скептиком.
- Функция потерь $L(\theta,\hat{\theta}) = 1 - e^{-(\theta-\hat{\theta})^2}$ ограничивается отрезком [0, 1] и отражает отсутствие интереса пользователя к достаточно удаленным от фактического значения оценкам. Она напоминает бинарную функцию потерь, но налагает не такой сильный штраф на оценки, близкие к фактическому значению параметра.
- Можно запрограммировать и сложные нелинейные функции потерь:

```
def loss(true_value, estimate):
    if estimate*true_value > 0:
        return abs(estimate - true_value)
    else:
        return abs(estimate)*(estimate - true_value)**2
```

- Еще один повседневный пример — функция потерь, используемая синоптиками. Синоптикам выгодно ошибаться в сторону завышения вероятности дождя.

Дело в том, что люди обычно считают, что лучше быть готовыми к дождю, даже если его не случится, чем промокнуть в противном случае. Поэтому синоптики искусственно завышают вероятность дождя и сообщают эти завышенные оценки — более выигрышные для них, чем незавышенные.

5.2.1. Функции потерь на практике

До сих пор мы работали при не слишком реалистичном допущении об известности фактического значения параметра. Разумеется, если фактическое значение параметра известно, то смысла в его оценке нет. Следовательно, функция потерь целесообразна, только когда фактическое значение параметра неизвестно.

При байесовском выводе обычно считается, что неизвестные параметры на самом деле представляют собой случайные переменные с априорным и апостериорным распределениями. Полученное из апостериорного распределения значение является одним из возможных фактических значений параметра. С его помощью можно вычислить потери при данной оценке. Поскольку у нас есть целое распределение (апостериорное) возможных значений неизвестного параметра, то нас должно интересовать скорее вычисление по такой оценке *ожидаемых потерь*. Эти ожидаемые потери — лучшая оценка фактических потерь, чем сравнение потерь по одной выборке из апостериорного распределения.

Во-первых, поговорим о том, что такое *байесовская точечная оценка* (bayesian point estimate). Современные системы и механизмы обработки данных не предназначены для получения в качестве входных данных апостериорных распределений. Кроме того, невежливо вручать распределение тому, кто просит у вас точечную оценку. В быту, сталкиваясь с неопределенностью, мы стараемся свести все многообразие возможных действий к одному. Аналогично необходимо свести апостериорное распределение к единому значению (или вектору в многомерном случае). Грамотно выбранное значение позволяет избежать недостатков частотных подходов, скрывающих неопределенность, и добиться более содержательных результатов. Выбранное из байесовского апостериорного распределения значение называется байесовской точечной оценкой.

Если $P(\theta|X)$ — апостериорное распределение θ после наблюдения данных X, то следующая функция трактуется как ожидаемые потери от *выбора* $\hat{\theta}$ *в качестве оценки* θ:

$$l(\hat{\theta}) = E_{\theta}\left[L(\theta, \hat{\theta}) \right].$$

Она также называется *риском* оценки $\hat{\theta}$. Индекс θ у символа математического ожидания отмечает тот факт, что θ — неизвестная (случайная) величина в математическом ожидании — нечто, что сложно сразу понять.

Всю главу 4 мы занимались приближенным вычислением математических ожиданий. По данным N выборок θ_i, $i = 1 \ldots N$ из апостериорного распределения и функции потерь L можно приближенно вычислить (по закону больших чисел) ожидаемые потери от использования оценки $\hat{\theta}$:

$$\frac{1}{N}\sum_{i=1}^{N} L(\theta_i, \hat{\theta}) \approx E_\theta\left[L(\theta, \hat{\theta})\right] = l(\hat{\theta}).$$

Отмечу, что оценка потерь с помощью математического ожидания требует больше информации из распределения, чем оценка MAP, которая, как вы помните, служит для нахождения лишь максимального значения распределения и игнорирует форму этого распределения. При игнорировании информации вы подвергаетесь высокому риску маловероятных событий вроде ураганов, а ваша оценка не учитывает степень вашей неуверенности в фактическом значении параметра.

Можно провести аналогию с частотными методами, которые обычно стремятся только минимизировать ошибку, не учитывая соответствующие потери. Кроме того, точность частотных методов практически наверняка никогда не будет абсолютной. Байесовские точечные оценки позволяют решить эту проблему за счет заблаговременного планирования: если уж оценка все равно окажется неточной, то лучше, чтобы она была неточной в нужную сторону.

5.2.2. Пример: оптимизация для раунда «Витрина» в викторине «Справедливая цена»

Если вас когда-нибудь выберут в качестве участника телевикторины «Справедливая цена» *(The Price Is Right)*, вам повезет, потому что сейчас я расскажу, как выбрать оптимальную цену в раунде «Витрина». Для тех, кто не знаком с правилами этого раунда, приведу их[1].

1. В раунде «Витрина» соревнуются два участника.
2. Каждому из них демонстрируется отдельный набор призов.
3. Участники должны сделать ставки на эти наборы призов.
4. Если ставка превышает фактическую цену, то поставивший ее участник лишается права на выигрыш.
5. Если ставка ниже фактической цены на не более чем $250, то победитель получает оба набора призов.

[1] На самом деле правила сложнее (см.: https://en.wikipedia.org/wiki/The_Price_Is_Right_(U.S._game_show)#The_Showcase), хотя для целей данной книги это неважно.

Сложность этой игры — найти оптимальный баланс своей неуверенности в значениях цен и поставить достаточно мало для того, чтобы не переборщить, но и достаточно много, чтобы ставка была близка к фактической цене.

Пусть у нас есть записи раундов «Витрина» из предыдущих эпизодов «Справедливой цены». У нас также есть априорная степень уверенности в распределении, которому подчиняется фактическая цена. Для простоты пусть это будет нормальное распределение:

$$\text{Фактическая цена} \sim \text{Normal}(\mu_p, \sigma_p).$$

Допустим для начала, что $\mu_p = 35\,000$, а $\sigma_p = 7500$.

Нам нужна модель того, как следует играть раунд «Витрина». У нас есть гипотеза относительно цены каждого приза из набора, но она может сильно отличаться от фактической цены (прибавьте это к дополнительному стрессу из-за направленных на вас камер, и вы поймете, почему некоторые ставки такие странные). Пусть априорная уверенность в цене каждого из призов также подчиняется нормальному распределению:

$$\text{Приз}_i \sim \text{Normal}(\mu_i, \sigma_i)\ ,\ i = 1, 2.$$

Байесовский анализ хорош именно возможностью задания предполагаемой разумной цены через параметр μ_i с одновременным выражением степени неуверенности через параметр σ_i. Для простоты будем считать, что в каждом наборе два приза, но при желании можно обобщить для произвольного количества призов. Фактическая цена набора призов при этом будет равна $\text{Приз}_1 + \text{Приз}_2 + \varepsilon$, где ε — некое рассогласование. Мы видели оба приза, у нас есть распределения степени уверенности относительно цен на них, и нам нужно узнать фактическую цену. Это можно сделать с помощью PyMC.

Конкретизируем некоторые значения. Пусть наблюдаемый набор включает следующие два приза.

1. Поездка в замечательный город Торонто, Канада!
2. Чудесный новый снегоочиститель!

У нас есть гипотезы относительно фактических цен этих объектов, но мы в них не слишком уверены. Выразить эту неуверенность можно с помощью параметров нормальных распределений:

$$\text{Снегоочиститель} \sim \text{Normal}(3000, 500);$$

$$\text{Торонто} \sim \text{Normal}(12000, 3000).$$

Например, я верю, что фактическая цена поездки в Торонто равна 12 000 долларов, причем существует вероятность 68,2 %, что цена отклоняется от этого значения

на одно стандартное отклонение; то есть я уверен в 68,2%-ной вероятности того, что цена поездки находится в диапазоне [9000, 15 000]. Эти априорные распределения наглядно показаны на рис. 5.1.

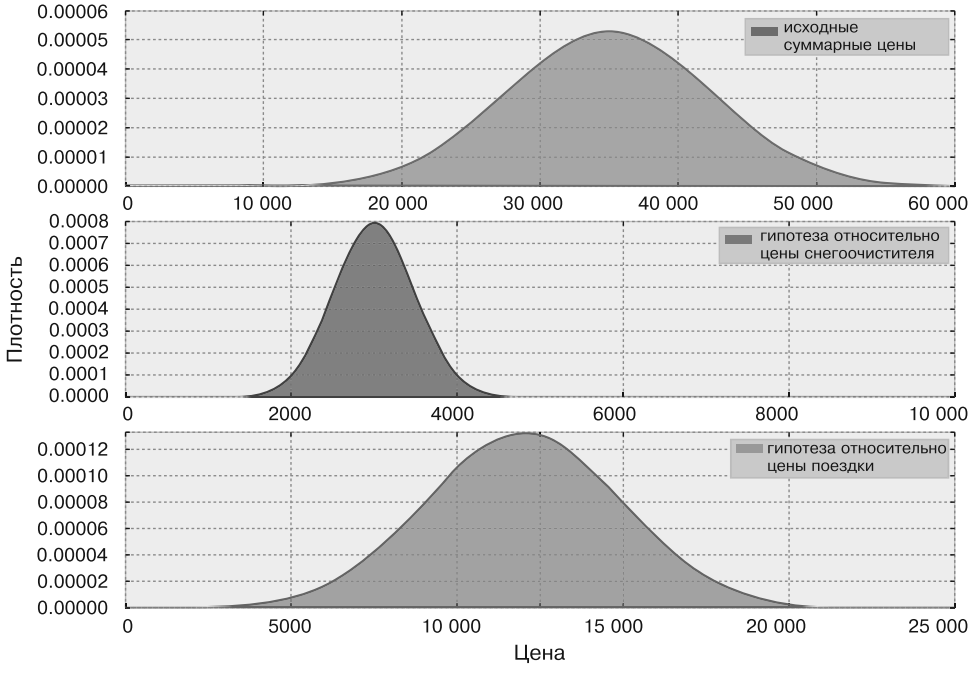

Рис. 5.1. Априорные распределения неизвестных величин: суммарная цена, цена снегоочистителя и цена поездки

Напишем код PyMC для вывода фактической цены набора, как показано на рис. 5.2.

```
%matplotlib inline
import scipy.stats as stats
from IPython.core.pylabtools import figsize
import numpy as np
import matplotlib.pyplot as plt
plt.rcParams['savefig.dpi'] = 300
plt.rcParams['figure.dpi'] = 300

figsize(12.5, 9)

norm_pdf = stats.norm.pdf

plt.subplot(311)
x = np.linspace(0, 60000, 200)
```

```
sp1 = plt.fill_between(x, 0, norm_pdf(x, 35000, 7500),
                       color="#348ABD", lw=3, alpha=0.6,
                       label=u"исходные суммарные цены")
p1 = plt.Rectangle((0, 0), 1, 1, fc=sp1.get_facecolor()[0])
plt.legend([p1], [sp1.get_label()])

plt.subplot(312)
x = np.linspace(0, 10000, 200)
sp2 = plt.fill_between(x, 0, norm_pdf(x, 3000, 500),
                       color="#A60628", lw=3, alpha=0.6,
                       label=u"гипотеза относительно цены снегоочистителя")

p2 = plt.Rectangle((0, 0), 1, 1, fc=sp2.get_facecolor()[0])
plt.legend([p2], [sp2.get_label()])

plt.subplot(313)
x = np.linspace(0, 25000, 200)
sp3 = plt.fill_between(x , 0, norm_pdf( x, 12000, 3000),
                       color="#7A68A6", lw=3, alpha=0.6,
                       label=u"гипотеза относительно цены поездки")
plt.autoscale(tight=True)
p3 = plt.Rectangle((0, 0), 1, 1, fc=sp3.get_facecolor()[0])
plt.title(u"Априорные распределения неизвестных величин: \
суммарная цена, цена снегоочистителя и цена поездки")
plt.legend([p3], [sp3.get_label()]);
plt.xlabel(u"Цена");
plt.ylabel(u"Плотность")
import pymc as pm

data_mu = [3e3, 12e3]

data_std = [5e2, 3e3]

mu_prior = 35e3
std_prior = 75e2
true_price = pm.Normal("true_price", mu_prior, 1.0 / std_prior ** 2)

prize_1 = pm.Normal("first_prize", data_mu[0], 1.0 / data_std[0] ** 2)
prize_2 = pm.Normal("second_prize", data_mu[1], 1.0 / data_std[1] ** 2)
price_estimate = prize_1 + prize_2

@pm.potential
def error(true_price=true_price, price_estimate=price_estimate):
    return pm.normal_like(true_price, price_estimate, 1 / (3e3) ** 2)

mcmc = pm.MCMC([true_price, prize_1, prize_2, price_estimate, error])
mcmc.sample(50000, 10000)

price_trace = mcmc.trace("true_price")[:]
```

```
[Output]:

[-----------------100%-----------------] 50000 of 50000 complete in 10.9 sec
```

```
figsize(12.5, 4)

import scipy.stats as stats

# Строим график априорного распределения.
x = np.linspace(5000, 40000)
plt.plot(x, stats.norm.pdf(x, 35000, 7500), c="k", lw=2,
         label=u"априорное распределение\n цены набора")

# Строим график апостериорного распределения, представленного выборками из MCMC.
_hist = plt.hist(price_trace, bins=35, normed=True, histtype="stepfilled")
plt.title(u"Апостериорное распределение оценки фактической цены")
plt.vlines(mu_prior, 0, 1.1*np.max(_hist[0]), label=u"среднее значение
           априорного распределения", linestyles="--")
plt.vlines(price_trace.mean(), 0, 1.1*np.max(_hist[0]), \
           label=u"среднее значение апостериорного распределения",
           linestyles="-.")
plt.legend(loc="upper left");
```

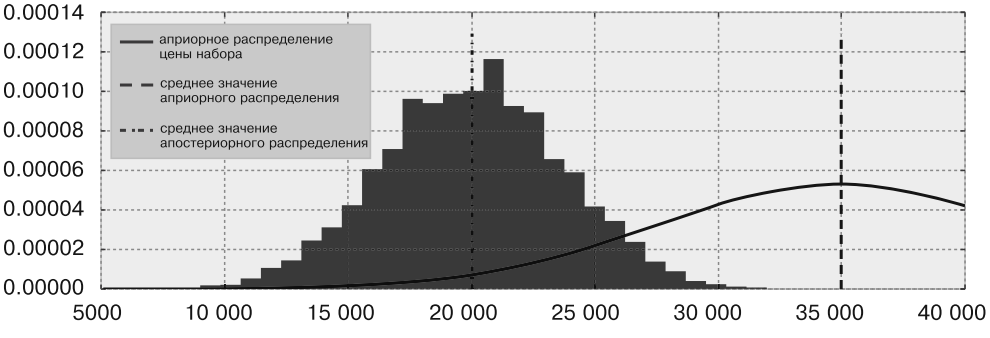

Рис. 5.2. Апостериорное распределение оценки фактической цены

Отмечу, что благодаря двум наблюдаемым ценам и последующим гипотезам (в том числе степени неопределенности) средняя оценка цены сдвинулась примерно на 15 000 от предыдущей средней цены.

Сторонник частотного подхода, увидев эти две цены, при той же априорной степени уверенности в стоимости этих призов сделал бы ставку, равную $\mu_1 + \mu_2 = 35\,000$, вне зависимости от неопределенности. В то же время сторонник *наивного байесовского подхода* просто поставил бы среднее значение апостериорного распределения. Но наша информация о возможных исходах на самом деле шире, и имеет смысл учесть ее в ставке. Воспользуемся функцией потерь для поиска *оптимальной* ставки (*оптимальной* относительно используемой функции потерь).

Как может выглядеть функция потерь участника викторины? Думаю, что примерно следующим образом:

```
def showcase_loss(guess, true_price, risk=80000):
    if true_price < guess:
        return risk
    elif abs(true_price - guess) <= 250:
        return -2 * np.abs(true_price)
    else:
        return np.abs(true_price - guess - 250)
```

Здесь `risk` — параметр, который определяет, сколько вы рискуете потерять, если ваша гипотеза окажется выше фактической цены. Возьмем для него, скажем, значение 80 000. Более низкое значение параметра `risk` говорит о том, что вас меньше волнует возможность превысить фактическую цену. Если наша ставка меньше фактической цены, причем разница между ними не превышает $250, то мы получаем оба приза (что мы моделируем как получение вдвое большей по сравнению с исходным призом суммы). Если же наша ставка меньше `true_price`, хотелось бы оказаться как можно ближе к `true_price`, поскольку функция потерь растет пропорционально расстоянию между нашей гипотезой и фактической ценой.

Мы будем вычислять *ожидаемые потери* для всех размеров ставки, а также пробовать различные значения параметра `risk`, чтобы выяснить, как он влияет на потери. Результаты показаны на рис. 5.3.

```
figsize(12.5, 7)
# Функция showdown_loss (NumPy)
def showdown_loss(guess, true_price, risk=80000):
        loss = np.zeros_like(true_price)
        ix = true_price < guess
        loss[~ix] = np.abs(guess - true_price[~ix])
        close_mask = [abs(true_price - guess) <= 250]
        loss[close_mask] = -2 * true_price[close_mask]
        loss[ix] = risk
        return loss

guesses = np.linspace(5000, 50000, 70)
risks = np.linspace(30000, 150000, 6)
expected_loss = lambda guess, risk: showdown_loss(guess, price_trace,
                                                  risk).mean()

for _p in risks:
    results = [expected_loss (_g, _p) for _g in guesses]
    plt.plot(guesses, results, label="%d"%_p)

plt.title(u"Ожидаемые потери при различных гипотезах и\
          \nразличных уровнях риска завышения цены")
plt.legend(loc="upper left", title=u"Параметр risk")
plt.xlabel(u"Ставка")
plt.ylabel(u"Ожидаемые потери")
plt.xlim(5000, 30000);
```

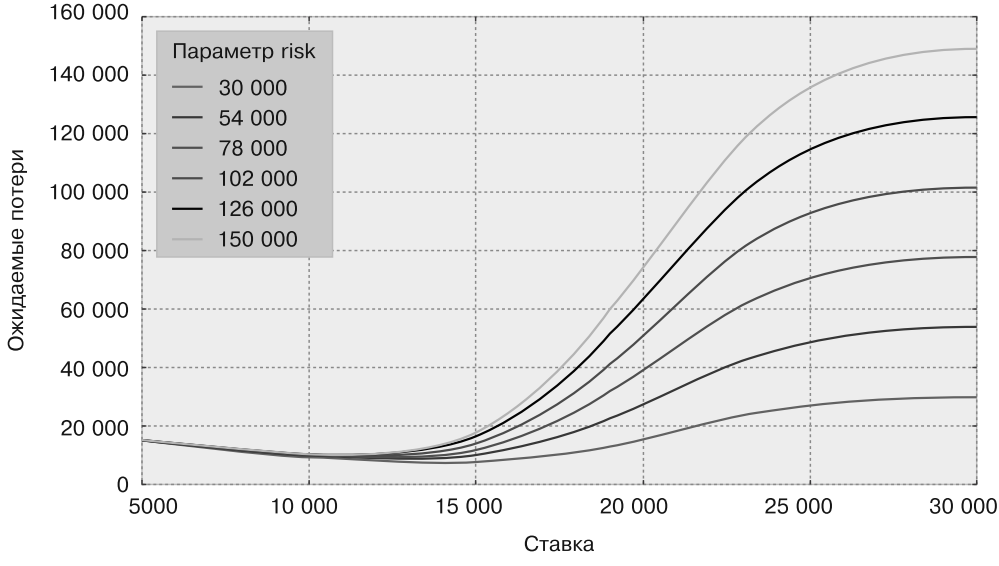

Рис. 5.3. Ожидаемые потери при различных гипотезах и различных уровнях риска завышения цены

Минимизация потерь. Имеет смысл выбрать оценку, которая минимизирует ожидаемые потери. Она соответствует точкам минимума кривых на предыдущем графике. Говоря более формализованным языком, необходимо минимизировать ожидаемые потери путем нахождения решения:

$$\arg\min_{\hat{\theta}} E_{\theta}\left[L(\theta,\hat{\theta})\right].$$

Минимум ожидаемых потерь называется *байесовской стратегией* (Bayesian action). Для ее нахождения можно воспользоваться утилитами оптимизации SciPy. Функция `fmin` из модуля `scipy.optimize` с помощью интеллектуального поиска находит минимум (отнюдь не обязательно *глобальный*) любой одно- или многомерной функции. В большинстве случаев результаты `fmin` должны вас вполне устроить.

На рис. 5.4 приведены результаты вычисления минимума потерь для примера с раундом «Витрина».

```
import scipy.optimize as sop

ax = plt.subplot(111)

for _p in risks:
    _color = ax._get_lines.color_cycle.next()
    _min_results = sop.fmin(expected_loss, 15000, args=(_p,),disp=False)
    _results = [expected_loss(_g, _p) for _g in guesses]
    plt.plot(guesses, _results , color=_color)
```

```
    plt.scatter(_min_results, 0, s=60, color=_color, label="%d"%_p)
    plt.vlines(_min_results, 0, 120000, color=_color, linestyles="--")
    print "минимум при риске %d: %.2f"%(_p, _min_results)

plt.title(u"Ожидаемые потери и байесовские стратегии при различных\n \
          гипотезах и различных уровнях риска завышения цены")
plt.legend(loc="upper left", scatterpoints=1,
          title=u"Байесовская стратегия при риске")
plt.xlabel(u"Гипотеза относительно цены")
plt.ylabel(u"Ожидаемые потери")
plt.xlim(7000, 30000)
plt.ylim(-1000, 80000);
```

```
[Output]:

минимум при риске 30000: 14189.08
минимум при риске 54000: 13236.61
минимум при риске 78000: 12771.73
минимум при риске 102000: 11540.84
минимум при риске 126000: 11534.79
минимум при риске 150000: 11265.78
```

```
[Output]:

(-1000, 80000)
```

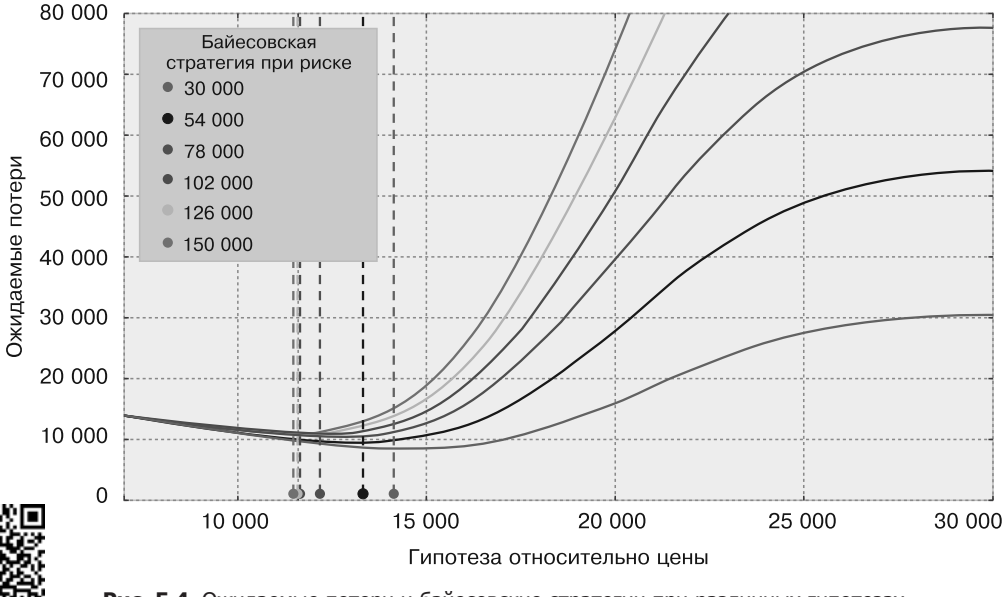

Рис. 5.4. Ожидаемые потери и байесовские стратегии при различных гипотезах и различных уровнях риска завышения цены

При снижении порога риска (когда нас меньше заботит возможность превышения цены) мы повышаем ставку, стараясь поближе подобраться к фактической цене. Любопытно наблюдать, насколько далек наш минимум потерь от апостериорного среднего, равного примерно 20 000.

Достаточно сказать, что при более высокой размерности увидеть своими глазами минимальные ожидаемые потери не получится. Именно поэтому нам необходима функция `fmin` из модуля SciPy.

Упрощенные способы. Для некоторых функций потерь байесовскую стратегию можно выразить аналитически. Перечислим часть из них.

- При среднеквадратичной функции потерь байесовская стратегия представляет собой среднее значение апостериорного распределения, то есть значение:

$$E_{\theta}[\theta]$$

  минимизирует выражение $E_{\theta}\left[(\theta-\hat{\theta})^2\right]$. В вычислительном смысле необходимо рассчитать среднее апостериорных выборок (см. описание закона больших чисел в главе 4).
- Поскольку медиана апостериорного распределения минимизирует ожидаемые абсолютные потери, выборочная медиана апостериорных выборок — вполне подходящее и довольно точное приближение фактической медианы.
- На самом деле можно показать, что оценка MAP — решение, подходящее в качестве бинарной функции потерь.

Наверное, вам уже стало понятнее, почему чаще всего при байесовском выводе используются приведенные выше функции потерь: при них не требуются никакие сложные оптимизации. К счастью, все сложные вычисления за нас выполняют машины.

5.3. Машинное обучение с помощью байесовских методов

Если частотные методы нацелены на достижение максимальной точности по всем вероятным параметрам, то при машинном обучении главное — наилучшее возможное *предсказание* по всем вероятным параметрам. Зачастую наша мера качества предсказания и оптимизируемая в частотных методах величина очень сильно различаются.

Например, линейная регрессия методом наименьших квадратов — простейший метод активного машинного обучения. *Активного* потому, что требует обучения, в то время как предсказание выборочного среднего — технически более простая задача, но обучения требует очень мало (или даже вообще не требует). В основе

функции потерь, определяющей коэффициенты регрессоров, лежит квадратичная ошибка. С другой стороны, если функция потерь предсказания (или обратная ей функция ценности) не отражает квадратичную ошибку, то кривая наименьших квадратов не будет оптимальной для предсказания функции потерь. При этом результаты предсказания могут оказаться недостаточно хорошими.

Нахождение байесовских стратегий эквивалентно нахождению параметров, при которых достигается экстремум не *точности параметров*, а *произвольной меры эффективности*. Как бы то ни было, необходимо определить, что такое в нашем понимании «эффективность» (функция потерь, ROC-кривая, AUC[1], точность и т. д.).

Следующие два примера подтверждают эти идеи. В первом представлена линейная модель, при которой можно выбрать для предсказания функцию потерь на основе метода наименьших квадратов или нестандартную функцию потерь с учетом возможных исходов. Второй пример — переделка проекта по исследованию данных с конкурса Kaggle. Функция потерь, используемая в нем для предсказаний, чрезвычайно сложна.

5.3.1. Пример: предсказание финансовых показателей

Пусть будущая отдача при данном курсе акций очень мала, скажем 0,01 (то есть 1 %). У нас есть модель для предсказания будущих курсов акций, а наш доход/потери напрямую зависят от наших действий на основе этих предсказаний. Как же нам оценить потери от этих предсказаний модели и последующих предсказаний? Функция потерь на основе квадратичной ошибки инвариантна относительно знака, а штраф для предсказанного значения –0,01 при ней равен штрафу от значения 0,03:

$$(0{,}01 - (-0{,}01))^2 = (0{,}01 - 0{,}03)^2 = 0{,}004.$$

Если вы сделаете ставку на основе предсказаний своей модели, то, вероятно, получите прибыль со значением 0,03 и потеряете деньги со значением –0,01, однако функция потерь этого не учитывает. Поэтому нам нужна более совершенная функция потерь, учитывающая *знак* предсказания и фактического значения. Создадим более подходящую для применения в финансовой сфере функцию потерь (рис. 5.5).

```
figsize(12.5, 4)
def stock_loss(true_return, yhat, alpha=100.):
    if true_return*yhat < 0:
        # Знаки противоположны — это плохо
        return alpha*yhat**2 - np.sign(true_return)*yhat \
                    + abs(true_return)
    else:
        return abs(true_return - yhat)

true_value = .05
```

[1] Площадь под ROC-кривой.

```
pred = np.linspace(-.04, .12, 75)

plt.plot(pred, [stock_loss(true_value, _p) for _p in pred], \
         label = u"Потери вследствие предсказания,\n
         если фактическое значение = 0.05", lw=3)
plt.vlines(0, 0, .25, linestyles="--")

plt.xlabel(u"Предсказание")
plt.ylabel(u"Потери" )
plt.xlim(-0.04, .12)
plt.ylim(0, 0.25)

true_value = -.02
plt.plot(pred, [stock_loss(true_value, _p) for _p in pred], alpha=0.6, \
         label=u"Потери вследствие предсказания,\n
         если фактическое значение = -0.02", lw=3)
plt.legend()
plt.title(u"Упущенная биржевая прибыль при фактическом значении = 0.05, -0.02" );
```

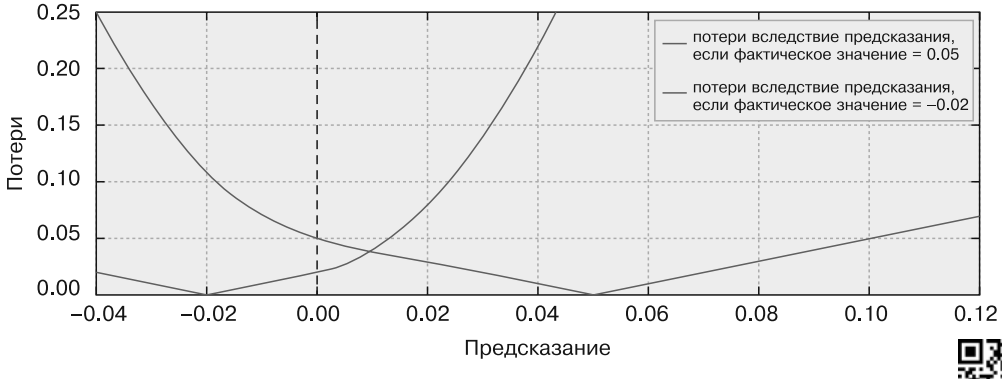

Рис. 5.5. Упущенная биржевая прибыль при фактическом значении = 0,05, –0,02

Обратите внимание на изменение формы кривой потерь при пересечении с предсказанием 0. Она отражает то, что пользователь вовсе не хочет ошибиться в знаке, причем ему особенно не хотелось совершить большую по модулю ошибку.

Почему пользователю важно абсолютное значение? Почему функция потерь не равна 0 в случае предсказания с правильным знаком? Ведь, если доход равен 0,01 при ставке несколько миллионов, нас это должно (более чем) устроить.

Дело в том, что финансовые учреждения одинаково (негативно) относятся как к рискам занижения (downside risk) — предсказаниям с большой ошибкой в неправильную сторону, так и к рискам завышения (upside risk) — предсказаниям с большой ошибкой в правильную сторону. Оба варианта считаются одинаково рискованными и не приветствуются. Следовательно, потери должны расти при удалении от фактической цены, причем их критичность чуть ниже в случае правильного знака.

Регрессию мы будем производить на основе торгового сигнала, что, как мне кажется, позволит достаточно хорошо предсказать будущую отдачу. Используемый набор данных — фиктивный, ведь почти все финансовые данные очень далеки от линейных. Построим график данных, а также линию наименьших квадратов (рис. 5.6).

```
# Код для создания фиктивных данных
N = 100
X = 0.025 * np.random.randn(N)
Y = 0.5 * X + 0.01 * np.random.randn(N)

ls_coef_ = np.cov(X, Y)[0,1]/np.var(X)
ls_intercept = Y.mean() - ls_coef_*X.mean()

plt.scatter(X, Y, c="k")
plt.xlabel(u"Торговый сигнал")
plt.ylabel(u"Отдача")
plt.title(u"Отдача по наблюдаемым данным относительно торгового сигнала")
plt.plot(X, ls_coef_ * X + ls_intercept, label="линия наименьших квадратов")
plt.xlim(X.min(), X.max())
plt.ylim(Y.min(), Y.max())
plt.legend(loc="upper left");
```

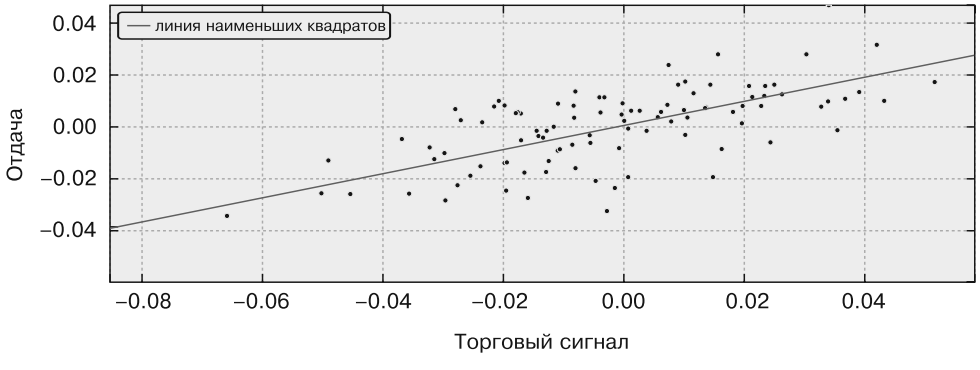

Рис. 5.6. Отдача по наблюдаемым данным относительно торгового сигнала

Произведем над этим набором данных простую линейную байесовскую регрессию. Наша модель имеет вид:

$$R = \alpha + \beta x + \varepsilon,$$

где α и β — неизвестные параметры, а $\varepsilon \sim \text{Normal}(0, 1 / \tau)$. В качестве априорных распределений β и α обычно используются нормальные. Назначим также априорное распределение для τ, такое, чтобы величина $\sigma = \frac{1}{\sqrt{\tau}}$ была равномерно распределена от 0 до 100 (что эквивалентно $\tau = 1 / \text{Uniform}(0, 100)^2$).

```
import pymc as pm
from pymc.Matplot import plot as mcplot

std = pm.Uniform("std", 0, 100, trace=False)

@pm.deterministic
def prec(U=std):
    return 1.0 / U **2

beta = pm.Normal("beta", 0, 0.0001)
alpha = pm.Normal("alpha", 0, 0.0001)

@pm.deterministic
def mean(X=X, alpha=alpha, beta=beta):
    return alpha + beta * X

obs = pm.Normal("obs", mean, prec, value=Y, observed=True)
mcmc = pm.MCMC([obs, beta, alpha, std, prec])

mcmc.sample(100000, 80000);
```

```
[Output]:

[-----------------100%-----------------] 100000 of 100000 complete in 23.2 sec
```

Для конкретного торгового сигнала — назовем его x — распределение возможной отдачи имеет вид:

$$R_i(x) = \alpha_i + \beta_i x + \varepsilon,$$

где $\varepsilon \sim \text{Normal}(0, 1 / \tau_i)$, а i — индекс для апостериорных выборок. Нам нужно найти решение для задачи оптимизации:

$$\arg\min_r E_{R(x)}\left[L(R(x), r)\right]$$

при заданной функции потерь; r — байесовская стратегия для торгового сигнала x. На рис. 5.7 приведены графики байесовских стратегий при различных торговых сигналах. Ничего не замечаете?

```
figsize(12.5, 6)
from scipy.optimize import fmin

def stock_loss(price, pred, coef=500):
    sol = np.zeros_like(price)
    ix = price*pred < 0
    sol[ix] = coef * pred **2 - np.sign(price[ix]) * pred + abs(price[ix])
    sol[~ix] = abs(price[~ix] - pred)
    return sol

tau_samples = mcmc.trace("prec")[:]
```

```
alpha_samples = mcmc.trace("alpha")[:]
beta_samples = mcmc.trace("beta")[:]

N = tau_samples.shape[0]

noise = 1. / np.sqrt(tau_samples) * np.random.randn(N)

possible_outcomes = lambda signal: alpha_samples + \
                                   beta_samples * signal + noise

opt_predictions = np.zeros(50)
trading_signals = np.linspace(X.min(), X.max(), 50)
for i, _signal in enumerate(trading_signals):
       _possible_outcomes = possible_outcomes(_signal)
       tomin = lambda pred: stock_loss(_possible_outcomes, pred).mean()
       opt_predictions[i] = fmin(tomin, 0, disp=False)

plt.xlabel(u"Торговый сигнал")
plt.ylabel(u"Предсказание")
plt.title(u"Предсказание по методу наименьших квадратов по сравнению\n
          с предсказанием с помощью байесовской стратегии" )
plt.plot(X, ls_coef_ * X + ls_intercept,
         label=u"предсказание метода наименьших квадратов")
plt.xlim(X.min(), X.max())
plt.plot(trading_signals, opt_predictions,
         label=u"предсказание байесовской стратегии")
plt.legend(loc="upper left");
```

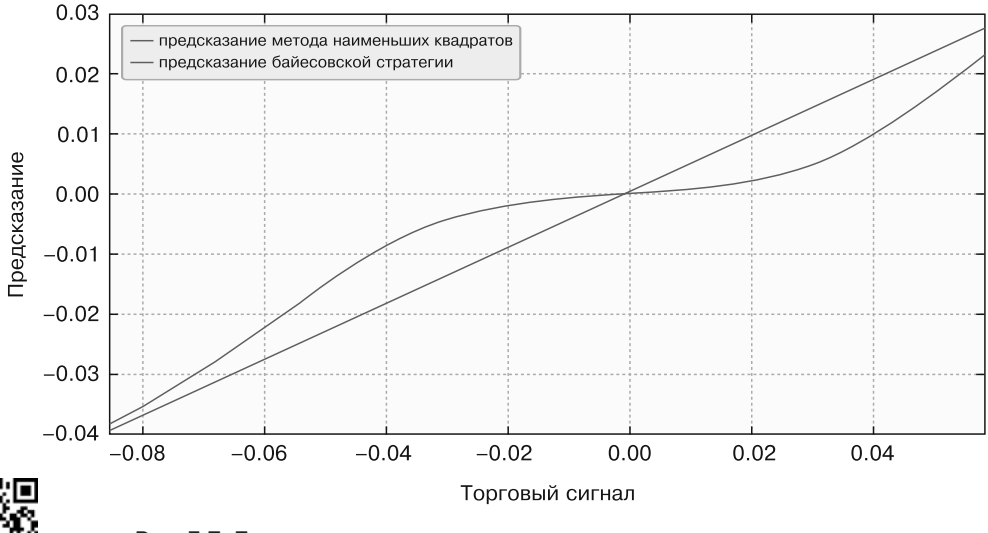

Рис. 5.7. Предсказание по методу наименьших квадратов по сравнению с предсказанием с помощью байесовской стратегии

В рис. 5.7 интересно то, что, когда сигнал близок к 0, а возможная отдача может быть как положительной, так и отрицательной, лучше всего (с учетом потерь) будет предсказать значение, близкое к 0; то есть не говорить решительно ни да, ни нет. Сказать что-то определенное можно лишь тогда, когда есть уверенность. Подобную модель, когда из-за неуверенности лучше не предпринимать действий, я называю *разреженным предсказанием* (sparse prediction). Сравните эту модель с предсказанием по методу наименьших квадратов, при котором очень редко (а то и вообще никогда) не будет предсказано значение 0.

В качестве подтверждения корректности модели можно рассматривать то, что по мере приближения сигнала к экстремуму (и повышения нашей уверенности в том, что отдача будет положительной/отрицательной) наше решение сходится с линией наименьших квадратов.

Модель разреженных предсказаний не стремится получить оптимальное предсказание в соответствии с критерием согласия в виде функции потерь на основе квадратичной ошибки. Это делает модель наименьших квадратов. Но модель разреженных предсказаний стремится найти оптимальное предсказание *относительно нашей функции потерь* `stock_loss`. Можно сформулировать это наоборот: модель наименьших квадратов не стремится предсказать лучшее значение (относительно определяемого функцией `stock_loss` понятия «предсказать»). Это относится к модели разреженных предсказаний. Модель наименьших квадратов стремится найти оптимальные параметры относительно функции потерь на основе квадратичной ошибки.

5.3.2. Пример: конкурс Kaggle по поиску темной материи

Я занялся изучением байесовских методов ради того, чтобы воссоздать решение, одержавшее победу на конкурсе Kaggle по поиску темной материи. С сайта конкурса [2]:

> «Вселенная состоит отнюдь не только из видимого глазом. В космосе присутствует разновидность материи, превышающая по количеству видимое нами вещество в семь раз, но мы понятия не имеем, что она из себя представляет. Известно только, что она не испускает и не поглощает свет, за что и получила название *темной материи* (Dark Matter).
>
> Такое колоссальное количество материи не могло остаться незамеченным учеными. Как было обнаружено, это вещество собирается вместе и образует крупные структуры — *гало темной материи* (Dark Matter Halos).
>
> Даже будучи темной, эта материя деформирует и искривляет пространство-время так, что проходящий поблизости с темной материей свет из галактик меняет свой путь. Вследствие этой деформации галактики визуально имеют эллиптическую форму».

Целью конкурса был поиск вероятных мест расположения темной материи. Победитель, Тим Сэлиманс (Tim Salimans), воспользовался байесовским выводом

для поиска наиболее вероятных местоположений гало (что интересно, занявший второе место участник также использовал байесовский вывод). С разрешения Тима я приведу здесь его решение [3].

1. Формируем априорное распределение местоположений гало $p(x)$, то есть формулируем наши ожидания относительно расположений гало, прежде чем взглянуть на данные.
2. Создаем вероятностную модель данных (наблюдаемых эксцентриситетов галактик) при заданных местоположениях гало темной материи: $p(e|x)$.
3. С помощью байесовского правила получаем апостериорное распределение местоположений гало, то есть обучаемся на данных приблизительно определять местоположения гало темной материи.
4. Минимизируем ожидаемые потери (с учетом апостериорного распределения) от предсказаний местоположений гало: $\hat{x} = \arg\min_{\text{предсказание}} E_{p(x|e)}\left[L(\text{предсказание}, x)\right]$, то есть находим оптимальные предсказания для заданной метрики ошибки.

Функция потерь в этой задаче очень непроста. Самые решительные читатели могут найти ее в файле `DarkWorldsMetric.py`. Хотя я советую вовсе его не читать, достаточно упомянуть, что функция потерь занимает около 160 строк кода — в одну строку такое не записать. Эта функция потерь пытается оценить точность предсказания, то есть евклидова расстояния, без какого-либо смещения. Больше подробностей можно найти на веб-странице конкурса.

Попробуем реализовать решение-победитель с помощью PyMC и наших знаний функций потерь.

5.3.3. Данные

Наш набор данных представляет собой 300 отдельных файлов, каждый из которых соответствует одному участку неба. В каждом файле (на участке неба) содержится от 300 до 720 галактик. Каждой галактике соответствуют координаты x и y в диапазоне от 0 до 4200, а также показатели эксцентриситета: e_1 и e_2. Узнать, что эти показатели означают, можно по адресу https://www.kaggle.com/c/DarkWorlds#an-introduction-to-ellipticity, но для нас это важно только с точки зрения визуализации. Таким образом, типичный участок неба выглядит примерно так, как показано на рис. 5.8.

```
from draw_sky2 import draw_sky

n_sky = 3 # выбираем файл/участок неба для изучения
data = np.genfromtxt("data/Train_Skies/Train_Skies/\
Training_Sky%d.csv"%(n_sky),
                     dtype=None,
                     skip_header=1,
                     delimiter=",",
                     usecols=[1,2,3,4])
print "Данные по галактикам на участке неба %d."%n_sky
print "координата_x, координата_y, e_1, e_2"
```

```
print data[:3]

fig = draw_sky(data)
plt.title(u"Координаты и эксцентриситеты галактик участка неба %d."%n_sky)
plt.xlabel(u"Координата $x$")
plt.ylabel(u"Координата $y$");
```

```
[Output]:

Данные по галактикам на участке неба 3.
координата_x, координата_y, e_1, e_2
[[ 1.62690000e+02 1.60006000e+03 1.14664000e-01 -1.90326000e-01]
[  2.27228000e+03 5.40040000e+02 6.23555000e-01  2.14979000e-01]
[  3.55364000e+03 2.69771000e+03 2.83527000e-01 -3.01870000e-01]]
```

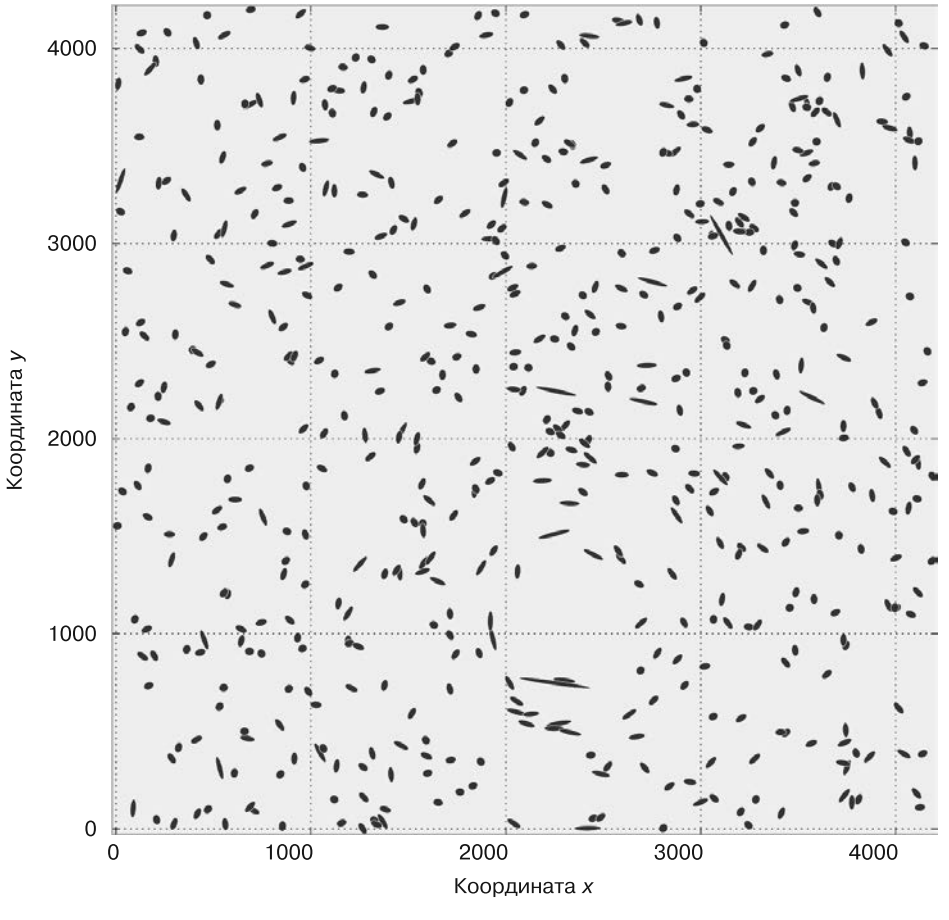

Рис. 5.8. Координаты и эксцентриситеты галактик участка неба 3

5.3.4. Априорные распределения

На каждом из участков неба есть от одного до трех гало темной материи. В описании решения Тима говорится, что его априорные распределения местоположений гало были равномерными, то есть:

$$x_i \sim \text{Uniform}(0, 4200);$$
$$y_i \sim \text{Uniform}(0, 4200),\ i = 1, 2, 3.$$

Тим и другие участники конкурса отмечали, что на большинстве участков неба было одно большое гало, остальные же (при наличии) были намного меньше. Большие гало за счет большей массы сильнее влияли на окружающие галактики. Он решил, что распределение масс больших гало должно соответствовать *логарифму* случайной переменной, равномерно распределенной между 40 и 180, то есть:

$$m_{\text{большое}} = \log \text{Uniform}(40, 180).$$

В коде PyMC это будет выглядеть следующим образом:

```
exp_mass_large = pm.Uniform("exp_mass_large", 40, 180)
@pm.deterministic
def mass_large(u = exp_mass_large):
    return np.log(u)
```

Массу меньших галактик Тим задал равной логарифму 20. Почему он не задал для них априорное распределение или не рассматривал их как неизвестные величины? Мне кажется, он принял такое решение для ускорения сходимости алгоритма. Оно не накладывает существенных ограничений, поскольку меньшие гало по определению слабее влияют на окружающие галактики.

Тим логично предположил, что эксцентриситет каждой галактики зависит от местоположения гало, расстояния между галактикой и гало, а также от массы гало. Следовательно, векторы эксцентриситетов галактик, e_i, являются переменными-потомками векторов местоположений гало (x, y), расстояния (которое мы формализуем далее) и масс гало.

Прочитав немало литературы и сообщений на форумах, Тим выяснил взаимосвязь между местоположениями гало и эксцентриситетами галактик. Он решил, что следующая формула взаимосвязи достаточно обоснованна:

$$e_i | (x, y) \sim \text{Normal}\Big(\sum_{j=\text{местоположения гало}} d_{i,j} m_j f(r_{i,j}), \sigma^2 \Big),$$

где $d_{i,j}$ — *тангенциальное направление* (направление, в котором гало j искривляет свет галактики i); m_j — масса гало j; $f(r_{i,j})$ — *убывающая функция* евклидова расстояния между гало j и галактикой i.

Функция Тима была определена как:

$$f(r_{i,j}) = \frac{1}{\min(r_{i,j}, 240)}$$

для больших гало и следующим образом для малых:

$$f(r_{i,j}) = \frac{1}{\min(r_{i,j}, 70)}.$$

Теперь наши наблюдения и неизвестные величины полностью связаны. Модель чрезвычайно проста, и Тим упоминал, что эта простота была намеренной во избежание переобучения модели.

5.3.5. Обучение и PyMC-реализация

Мы запускаем байесовскую модель для каждого из участков неба, чтобы получить апостериорные распределения координат гало, — известные местоположения гало мы игнорируем. Это несколько отличается от, наверное, более традиционных подходов к решению задач Kaggle: модель не использует данные с других участков неба или известные местоположения гало. Это вовсе не значит, что остальные данные не нужны; на самом деле модель была создана путем сравнения различных участков неба.

```
def euclidean_distance(x, y):
    return np.sqrt(((x - y) **2).sum(axis=1))

def f_distance(gxy_pos, halo_pos, c):
   # ..._position должен быть двумерным массивом Numpy.
   return np.maximum(euclidean_distance(gxy_pos, halo_pos), c)[:,None]

def tangential_distance(glxy_position, halo_position):
   # ..._position должен быть двумерным массивом Numpy.
   delta = glxy_position - halo_position
   t = (2*np.arctan(delta[:,1]/delta[:,0]))[:,None]
   return np.concatenate([-np.cos(t), -np.sin(t)], axis=1)

import pymc as pm

# Задаем порядок массы гало.
mass_large = pm.Uniform("mass_large", 40, 180, trace=False)

# Задаем исходные априорные местоположения гало в виде двумерного
# равномерного распределения.
halo_position = pm.Uniform("halo_position", 0, 4200, size=(1,2))

@pm.deterministic
```

```
def mean(mass=mass_large, h_pos=halo_position, glx_pos=data[:,:2]):
    return mass/f_distance(glx_pos, h_pos, 240)*\
            tangential_distance(glx_pos, h_pos)

ellpty = pm.Normal("ellipticity", mean, 1./0.05, observed=True, value=data[:,2:] )
mcmc = pm.MCMC([ellpty, mean, halo_position, mass_large])
map_ = pm.MAP([ellpty, mean, halo_position, mass_large])
map_.fit()
mcmc.sample(200000, 140000, 3)
```

```
[Output]:

[****************100%*****************] 200000 of 200000 complete
```

Построим график апостериорного распределения (просто диаграмму рассеяния апостериорного распределения, которую, однако, можно визуализировать в виде карты интенсивности) (рис. 5.9). Как вы можете видеть из графика, красное пятно (в центре. — *Примеч. ред.*) отмечает апостериорное распределение вероятности над местоположением гало.

```
t = mcmc.trace("halo_position")[:].reshape( 20000,2)

fig = draw_sky(data)
plt.title(u"Координаты и эксцентриситеты галактик участка неба %d."%n_sky)
plt.xlabel(u"Координата $x$")
plt.ylabel(u"Координата $y$");
plt.scatter(t[:,0], t[:,1], alpha=0.015, c="r")
plt.xlim(0, 4200)
plt.ylim(0, 4200);
```

Наиболее вероятное местоположение выглядит на этом графике как смертельная рана.

Каждому участку неба соответствует и еще по одной точке данных из файла `Training_halos.csv`, содержащего до трех местоположений гало темной материи каждого участка. Например, вот местоположения гало для участка ночного неба, на котором мы производили обучение:

```
halo_data = np.genfromtxt("data/Training_halos.csv",
                      delimiter=",",
                      usecols=[1,2,3,4,5,6,7,8,9],
                      skip_header=1)
print halo_data[n_sky]
```

```
[Output]:

[ 3.00000000e+00 2.78145000e+03 1.40691000e+03 3.08163000e+03
  1.15611000e+03 2.28474000e+03 3.19597000e+03 1.80916000e+03
  8.45180000e+02]
```

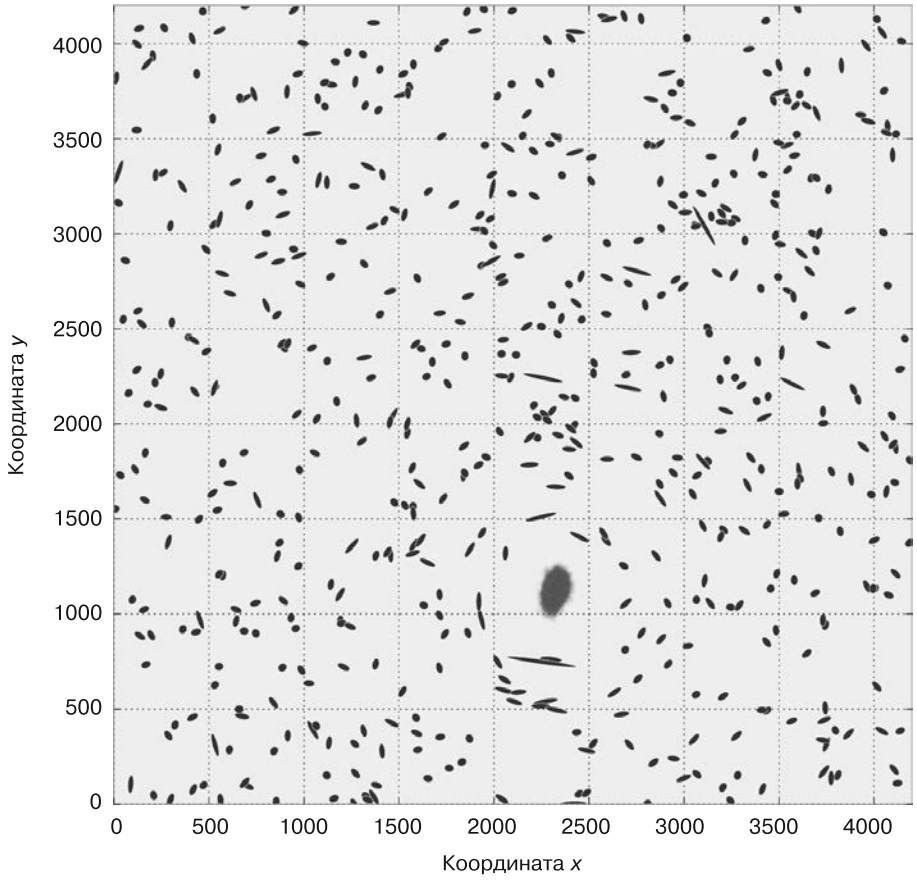

Рис. 5.9. Координаты и эксцентриситеты галактик участка неба 3

Третий и четвертый столбцы отражают фактические координаты x и y гало. Похоже, что байесовский метод весьма точно определил местоположение гало, отмеченное черной точкой на рис. 5.10.

```
fig = draw_sky(data)
plt.title(u"Координаты и эксцентриситеты галактик участка неба %d."%n_sky)
plt.xlabel(u"Координата $x$")
plt.ylabel(u"Координата $y$");
plt.scatter(t[:,0], t[:,1], alpha=0.015, c="r")
plt.scatter(halo_data[n_sky-1][3], halo_data[n_sky-1][4],
            label=u"фактическое местоположение гало",
            c="k", s=70)
plt.legend(scatterpoints=1, loc="lower left")
plt.xlim(0, 4200)
plt.ylim(0, 4200);
```

```
print "Фактическое местоположение гало:", halo_data[n_sky][3], halo_data[n_sky][4]
```

```
[Output]:

Фактическое местоположение гало: 1408.61 1685.86
```

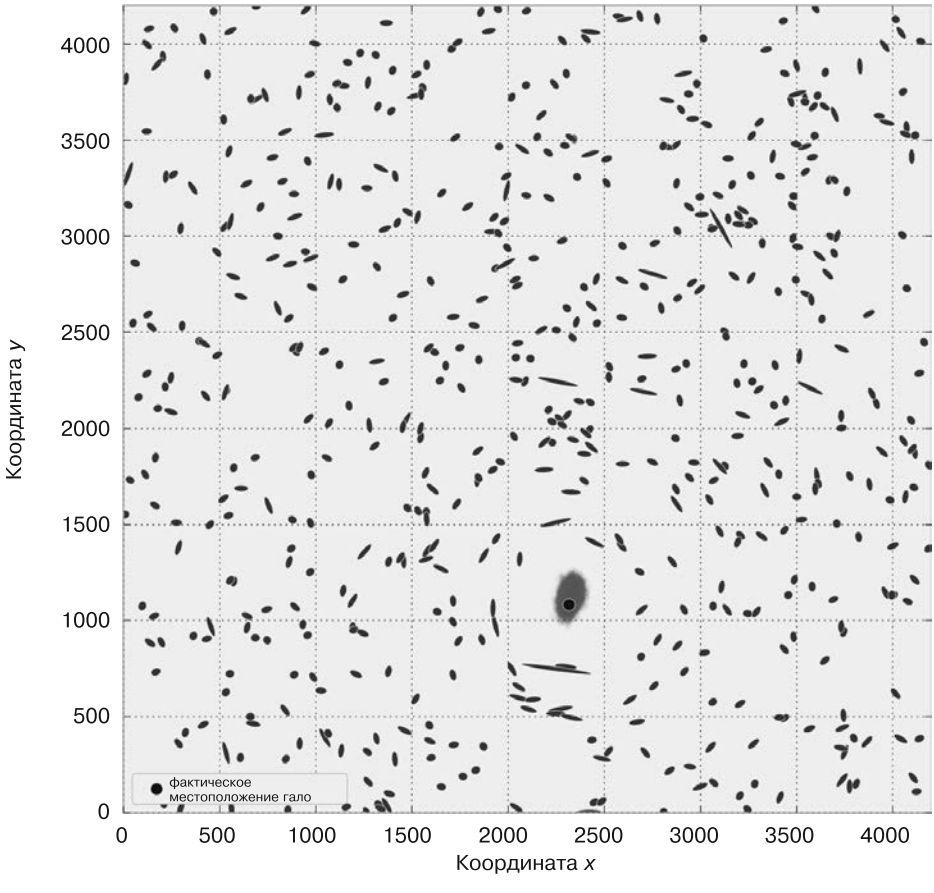

Рис. 5.10. Координаты и эксцентриситеты галактик участка неба 3

Отлично. Следующий шаг: оптимизация местоположения на основе функции потерь. Самая наивная методика — просто выбрать среднее значение:

```
mean_posterior = t.mean(axis=0).reshape(1,2)
print mean_posterior
```

```
[Output]:

[[ 2324.07677813 1122.47097816]]
```

```
from DarkWorldsMetric import main_score

_halo_data = halo_data[n_sky-1]

nhalo_all = _halo_data[0].reshape(1,1)
x_true_all = _halo_data[3].reshape(1,1)
y_true_all = _halo_data[4].reshape(1,1)
x_ref_all = _halo_data[1].reshape(1,1)
y_ref_all = _halo_data[2].reshape(1,1)
sky_prediction = mean_posterior

print "При использовании среднего значения:"

main_score(nhalo_all, x_true_all, y_true_all, \
           x_ref_all, y_ref_all, sky_prediction)

# А в худшем случае?
print
random_guess = np.random.randint(0, 4200, size=(1,2))
print "При использовании случайного местоположения:", random_guess

main_score(nhalo_all, x_true_all, y_true_all, \
           x_ref_all, y_ref_all, random_guess)
print
```

```
[Output]:

При использовании среднего значения:
Your average distance in pixels away from the true halo is 31.1499201664
Your average angular vector is 1.0
Your score for the training data is 1.03114992017

При использовании случайного местоположения: [[2755 53]]
Your average distance in pixels away from the true halo is 1773.42717812
Your average angular vector is 1.0
Your score for the training data is 2.77342717812
```

Гипотеза довольно неплохая, не очень далеко от фактического местоположения, однако она не учитывает имеющуюся у нас функцию потерь. Нам нужно также дописать код для учета двух дополнительных, *меньших* гало. Создадим функцию для автоматизации нашего PyMC-алгоритма.

```
from pymc.Matplot import plot as mcplot

def halo_posteriors(n_halos_in_sky, galaxy_data,
                    samples = 5e5, burn_in = 34e4, thin = 4):

    # Задаем порядок массы гало.

    mass_large = pm.Uniform("mass_large", 40, 180)

    mass_small_1 = 20
```

```
    mass_small_2 = 20

    masses = np.array([mass_large,mass_small_1, mass_small_2],
                      dtype=object)

    # Задаем исходные априорные местоположения гало в виде
    # двумерного равномерного распределения.
    halo_positions = pm.Uniform("halo_positions", 0, 4200,
                     size=(n_halos_in_sky,2))
    fdist_constants = np.array([240, 70, 70])

    @pm.deterministic
    def mean(mass=masses, h_pos=halo_positions, glx_pos=data[:,:2],
             n_halos_in_sky = n_halos_in_sky):

        _sum = 0
        for i in range(n_halos_in_sky):
            _sum += mass[i] / f_distance( glx_pos,h_pos[i, :],
                fdist_constants[i])*\
                    tangential_distance( glx_pos, h_pos[i, :])

        return _sum

    ellpty = pm.Normal("ellipticity", mean, 1. / 0.05, observed=True,
                       value = data[:,2:])

    map_ = pm.MAP([ellpty, mean, halo_positions, mass_large])
    map_.fit(method="fmin_powell")

    mcmc = pm.MCMC([ellpty, mean, halo_positions, mass_large])
    mcmc.sample(samples, burn_in, thin)
    return mcmc.trace("halo_positions")[:]

n_sky =215
data = np.genfromtxt("data/Train_Skies/Train_Skies/\
Training_Sky%d.csv"%(n_sky),
                     dtype=None,
                     skip_header=1,
                     delimiter=",",
                     usecols=[1,2,3,4])

# В этом файле содержится 3 гало.
samples = 10.5e5
traces = halo_posteriors(3, data, samples=samples,
                                  burn_in=9.5e5,
                                  thin=10)
```

```
[Output]:

[****************100%*****************] 1050000 of 1050000 complete
```

```
fig = draw_sky(data)
plt.title(u"Координаты и эксцентриситеты галактик, а также гало участка неба
%d."%n_sky)
plt.xlabel(u"Координата $x$")
plt.ylabel(u"Координата $y$");

colors = ["#467821", "#A60628", "#7A68A6"]

for i in range(traces.shape[1]):
    plt.scatter(traces[:, i, 0], traces[:, i, 1], c=colors[i], alpha=0.02)

for i in range(traces.shape[1]):
    plt.scatter(halo_data[n_sky-1][3 + 2 * i],
        halo_data[n_sky-1][4 + 2 * i],
            label=u"Фактическое местоположение гало", c="k", s=90)

plt.xlim(0, 4200)
plt.ylim(0, 4200);
```

```
[Output]:

(0, 4200)
```

Как вы можете видеть на рис. 5.11, получилось неплохо, хотя, чтобы (более или менее) сойтись, системе понадобилось немало времени. Шаг оптимизации будет выглядеть примерно следующим образом.

```
_halo_data = halo_data[n_sky-1]
print traces.shape

mean_posterior = traces.mean(axis=0).reshape(1,4)
print mean_posterior

nhalo_all = _halo_data[0].reshape(1,1)
x_true_all = _halo_data[3].reshape(1,1)
y_true_all = _halo_data[4].reshape(1,1)
x_ref_all = _halo_data[1].reshape(1,1)
y_ref_all = _halo_data[2].reshape(1,1)
sky_prediction = mean_posterior

print "При использовании среднего значения:"
main_score([1], x_true_all, y_true_all, \
           x_ref_all, y_ref_all, sky_prediction)

# А в худшем случае?
print
random_guess = np.random.randint(0, 4200, size=(1,2))
print "При использовании случайного местоположения:", random_guess
main_score([1], x_true_all, y_true_all, x_ref_all, y_ref_all, random_guess)
print
```

```
[Output]:

(10000L, 2L, 2L)
[[ 48.55499317 1675.79569424 1876.46951857 3265.85341193]]
При использовании среднего значения:
Your average distance in pixels away from the true halo is 37.3993004245
Your average angular vector is 1.0
Your score for the training data is 1.03739930042

При использовании случайного местоположения: [[2930 4138]]
Your average distance in pixels away from the true halo is 3756.54446887
Your average angular vector is 1.0
Your score for the training data is 4.75654446887
```

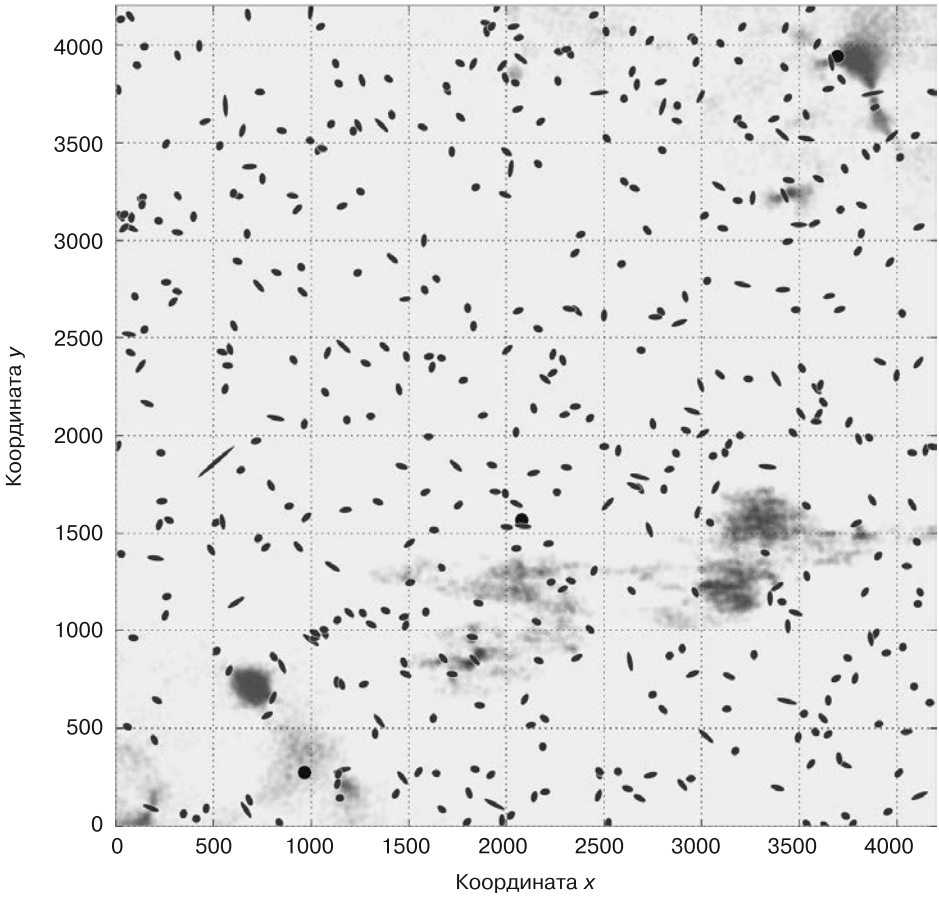

Рис. 5.11. Координаты и эксцентриситеты галактик, а также гало участка неба 215

5.4. Выводы

Функции потерь — одни из самых интересных элементов статистики. Они позволяют напрямую связать вывод с предметной областью задачи. Кстати, я *не* упомянул, что функция потерь — еще одна степень свободы в общей модели. Как мы видели в этой главе, это положительный фактор — функции потерь можно использовать очень эффективно. Но это может оказаться и недостатком. В крайнем случае работающий с конкретной задачей специалист может поменять используемую функцию потерь, если результаты оказываются не такими, как хотелось бы. Поэтому лучше задавать функцию потерь в самом начале анализа, причем выводить ее логичным и открытым способом.

5.5. Библиография

1. *Read C.* Logic: Deductive and Inductive. London: Simkin, Marshall, 1920. P. vi.
2. Observing Dark Worlds. https://www.kaggle.com/c/DarkWorlds.
3. *Salimans T.* Observing Dark Worlds. Tim Salimans on Data Analysis. http://timsalimans.com/observing-dark-worlds/.

6 Расставляем приоритеты

6.1. Введение

Эта глава посвящена наиболее неоднозначной и часто обсуждаемой области методологии байесовского вывода: выбору подходящего априорного распределения. Кроме того, я покажу, как по мере роста набора данных меняется влияние априорного распределения, и расскажу об интересных взаимосвязях между априорными распределениями и штрафами в линейной регрессии.

На протяжении этой книги мы в основном игнорировали выбор априорных распределений. И совершенно напрасно, ведь с их помощью можно очень многое выразить, хотя выбирать их следует с осторожностью. Особенно если мы хотим быть объективными, то есть не выражать что-либо посредством априорных распределений нашей личной уверенности.

6.2. Субъективные и объективные априорные распределения

Байесовские априорные распределения можно разбить на две категории: **объективные априорные распределения**, при которых наибольший вклад в апостериорное распределение вносят данные, и **субъективные априорные распределения**, позволяющие пользователю выразить в априорном распределении свои собственные взгляды.

6.2.1. Объективные априорные распределения

Мы уже сталкивались с примерами объективных априорных распределений, включая **плоское априорное распределение** (flat prior): равномерное распределение по всему диапазону значений неизвестных величин. Использование плоского априорного распределения подразумевает, что всем возможным значениям придается одинаковый вес. Выбор такого априорного распределения реализует

так называемый **принцип безразличия** (principle of indifference), гласящий, что нет никаких априорных причин предпочитать одно значение другому. Называть плоское априорное распределение по ограниченному пространству объективным было бы не совсем правильно, хотя кажется, что принцип остается тем же. Если при биномиальной модели известно, что p превышает 0,5, то нельзя сказать, что Uniform(0,5, 1) — объективное априорное распределение (поскольку при его выборе использовались внешние знания), хотя распределение, конечно, «плоское» в рамках отрезка [0,5, 1]. Плоское априорное распределение должно быть плоским над всем диапазоном возможных значений, включая отрезок [0, 0,5].

Существуют и другие, не столь очевидные примеры объективных априорных распределений, помимо плоского, которые содержат важные характеристики, отражающие объективность. Пока скажу лишь, что очень редко объективное априорное распределение *действительно* объективно. Мы поговорим об этом позднее.

6.2.2. Субъективные априорные распределения

С другой стороны, если придать больше вероятностного веса определенным областям априорного распределения и меньше — остальным областям, то вывод будет смещаться в сторону значений параметров из областей с большим вероятностным весом. Подобное называется *субъективным* (или *информативным*) априорным распределением.

Приведенное на рис. 6.1 субъективное априорное распределение отражает уверенность в том, что неизвестная величина близка к 0,5, а не к краям интервала. Объективное априорное распределение эту уверенность игнорирует.

```
%matplotlib inline
import numpy as np
from IPython.core.pylabtools import figsize
import matplotlib.pyplot as plt
import scipy.stats as stats
plt.rcParams['savefig.dpi'] = 300
plt.rcParams['figure.dpi'] = 300

figsize(12.5,3)
colors = ["#348ABD", "#A60628", "#7A68A6", "#467821"]

x = np.linspace(0,1)
y1, y2 = stats.beta.pdf(x, 1, 1), stats.beta.pdf(x, 10, 10)

p = plt.plot(x, y1,
      label=u'объективное априорное \nраспределение (неинформативное,\n
            отражающее принцип безразличия)')
plt.fill_between(x, 0, y1, color=p[0].get_color(), alpha=0.3)

p = plt.plot(x, y2,
      label=u'субъективное априорное \nраспределение (информативное)')
```

```
plt.fill_between(x, 0, y2, color=p[0].get_color(), alpha=0.3)

p = plt.plot(x[25:], 2*np.ones(25), label=u"еще одно субъективное \n
             априорное распределение")
plt.fill_between(x[25:], 0, 2, color=p[0].get_color(), alpha=0.3)

plt.ylim(0, 4)

plt.ylim(0, 4)
leg = plt.legend(loc="upper left")
leg.get_frame().set_alpha(0.4)
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.title(u"Сравнение объективных и субъективных априорных распределений \
          для неизвестной вероятности");
```

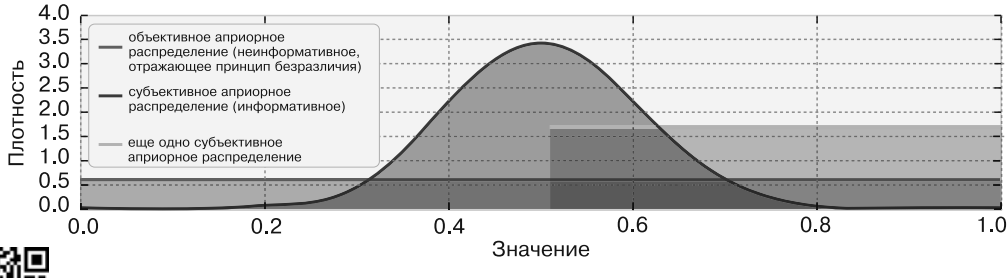

Рис. 6.1. Сравнение объективных и субъективных априорных распределений для неизвестной вероятности

Выбор субъективного априорного распределения не всегда означает использование субъективного мнения специалиста. Чаще всего субъективное априорное распределение когда-то было апостериорным распределением для предыдущей задачи, а теперь специалист просто обновляет его на основе новых данных. Субъективное априорное распределение может также использоваться для включения в модель знаний о предметной области. Мы рассмотрим далее примеры этих двух ситуаций.

6.2.3. Выбираем, выбираем...

Что выбрать: объективное или субъективное априорное распределение, в основном зависит от решаемой задачи, но существует несколько случаев, когда один из этих вариантов предпочтительнее. При научных исследованиях выбор объективного априорного распределения очевиден, поскольку позволяет избежать систематических погрешностей в результатах. Объективное априорное распределение должно

быть таким, чтобы устраивать исследователей с различными взглядами на объект исследований.

Рассмотрим более утрированную ситуацию. Допустим, что табачная компания опубликовала отчет, основанный на байесовской методике, подвергающий сомнению 60-летние медицинские исследования по вопросу курения табака. Поверите ли вы этим результатам? Вряд ли. Вероятно, исследователи выбрали субъективное априорное распределение, за счет чего результаты сильно сместились в выгодную им сторону.

К сожалению, выбор объективного априорного распределения зачастую существенно сложнее выбора плоского априорного распределения, и даже сегодня эта задача не вполне решена. Проблема с «наивным» выбором равномерного априорного распределения состоит в опасности возникновения принципиально не решаемых проблем. Некоторые из них хорошо известны и описаны в литературе, и далее мы увидим пример такой проблемы.

Не забывайте, что выбор априорного распределения, субъективного или объективного, остается частью процесса моделирования. Процитируем Гелмана (Gelman) [1]:

> «После обучения модели следует посмотреть на апостериорное распределение и решить для себя, имеет ли оно смысл. Если апостериорное распределение не имеет смысла, значит существуют не включенные в модель дополнительные априорные знания, которые противоречат используемому априорному распределению. В подобном случае следует вернуться к предыдущему шагу и согласовать априорное распределение с этим внешним знанием».

Если апостериорное распределение представляется вам бессмысленным, значит у вас есть какое-то представление о том, как апостериорное распределение *должно* выглядеть (не путайте с тем, как вам *хотелось бы*, чтобы оно выглядело), а значит, текущее априорное распределение включает не всю априорную информацию и его нужно усовершенствовать. На этом этапе можно отбросить текущее априорное распределение и выбрать такое, которое лучше отражает *всю* априорную информацию.

Гелман [2] считает, что использование равномерного распределения с широкими границами в качестве объективного априорного распределения — хороший выбор. Однако применять такие распределения следует с осторожностью, поскольку у интуитивно совершенно не подходящих точек может оказываться слишком большая априорная вероятность. Задайте себе вопрос: действительно ли вы считаете, что неизвестная величина *может* принимать очень большое значение? Зачастую величины естественным образом стремятся к 0. Возможно, для ваших целей лучше

подойдет нормальная случайная переменная с большой дисперсией (маленькой точностью) или — в строго положительном (отрицательном) случае — экспоненциальная переменная с «толстохвостым» распределением.

6.2.4. Эмпирическая байесовская оценка

Эмпирический байесовский подход сочетает в себе частотный и байесовский вывод. Как уже упоминалось, практически для каждой задачи вывода существует как байесовское, так и частотное решение. Основная разница между ними состоит в наличии у байесовских методов априорного распределения с гиперпараметрами α и τ, которое отсутствует у частотных методов. Эмпирический байесовский подход объединяет оба подхода путем выбора α и τ с помощью частотных методов с последующим решением исходной задачи байесовскими методами.

Приведу очень простой пример. Пусть нам нужно оценить параметр μ нормального распределения с $\sigma = 5$. Поскольку μ может принимать любое вещественное значение, в качестве априорного распределения для μ можно воспользоваться нормальным распределением. Далее нам нужно выбрать гиперпараметры априорного распределения, которые мы обозначим (μ_p, σ_p^2). Параметр σ_p^2 можно выбрать, исходя из степени неопределенности. Что же касается μ_p, то вариантов два.

1. В эмпирическом байесовском подходе рекомендуется использовать эмпирическое выборочное среднее для центрирования априорного распределения относительно наблюдаемого эмпирического среднего:

$$\mu_p = \frac{1}{N}\sum_{i=0}^{N} X_i.$$

2. При обычном байесовском подходе рекомендуется воспользоваться априорными знаниями или более объективным априорным распределением (средним значением 0 и «толстым» стандартным отклонением).

Некоторые рассматривают эмпирический байесовский подход как лишь наполовину объективный, в отличие от объективного байесовского вывода. Несмотря на то что мы сами выбираем априорную модель (привнося субъективность), параметры полностью определяются данными (поэтому он объективен).

Мне кажется, что использовать эмпирический байесовский подход — значит учитывать данные дважды. То есть один раз данные учитываются в априорном распределении, и в итоге результаты будут смещаться в сторону наблюдаемых данных, а потом еще раз — в механизме вывода MCMC. При подобном двойном учете легко недооценить фактическую неопределенность. Для минимизации тако-

го двойного учета я предложил бы вам использовать эмпирический байесовский подход только при наличии большого массива наблюдений, в противном случае влияние априорного распределения окажется слишком сильным. Я рекомендую также по возможности поддерживать высокую степень неопределенности (путем задания большого значения параметра σ_p^2 и т. п.).

Кроме того, эмпирический байесовский подход нарушает саму философию байесовского вывода. Он не соответствует традиционной байесовской последовательности шагов:

априорное распределение ⇒ *наблюдаемые данные* ⇒ *апостериорное распределение*,

используя взамен следующий алгоритм:

наблюдаемые данные ⇒ *априорное распределение* ⇒ *наблюдаемые данные* ⇒ *апостериорное распределение*.

В идеале необходимо задать все априорные распределения до наблюдения данных, чтобы данные не влияли на наши априорные мнения (см. исследования Дэниела Канемана (Daniel Kahneman) и др., посвященные такому явлению, как эффект привязки (anchoring)).

6.3. Некоторые полезные априорные распределения

Далее мы пройдемся по некоторым распределениям, часто встречающимся в байесовском анализе.

6.3.1. Гамма-распределение

Гамма-распределенная случайная переменная, обозначаемая $X \sim \text{Gamma}(\alpha, \beta)$, представляет собой случайную переменную, определенную на области положительных вещественных чисел. Фактически это обобщение экспоненциальной случайной переменной, а именно:

$$\text{Exp}(\beta) \sim \text{Gamma}(1, \beta).$$

Благодаря дополнительному параметру повышается гибкость функции плотности распределения вероятности, что дает возможность работающему с ней специалисту точнее выражать свои субъективные априорные представления. Функция

плотности для гамма-распределенной (Gamma(α, β)) случайной переменной выглядит следующим образом:

$$f(x \mid \alpha, \beta) = \frac{\beta^{\alpha} x^{\alpha-1} e^{-\beta x}}{\Gamma(\alpha)},$$

где $\Gamma(\alpha)$ — гамма-функция. Построим график гамма-распределения при различных значениях (α, β) (рис. 6.2):

```
figsize(12.5, 5)
gamma = stats.gamma

parameters = [(1, 0.5), (9, 2), (3, 0.5), (7, 0.5)]
x = np.linspace(0.001, 20, 150)
for alpha, beta in parameters:
    y = gamma.pdf(x, alpha, scale=1./beta)
    lines = plt.plot(x, y, label="(%.1f,%.1f)"%(alpha,beta), lw=3)
    plt.fill_between(x, 0, y, alpha=0.2, color=lines[0].get_color())
    plt.autoscale(tight=True)

plt.legend(title=ur"$\alpha, \beta$ - параметры")
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.title(ur"Гамма-распределение при различных значениях $\alpha$ и $\beta$");
```

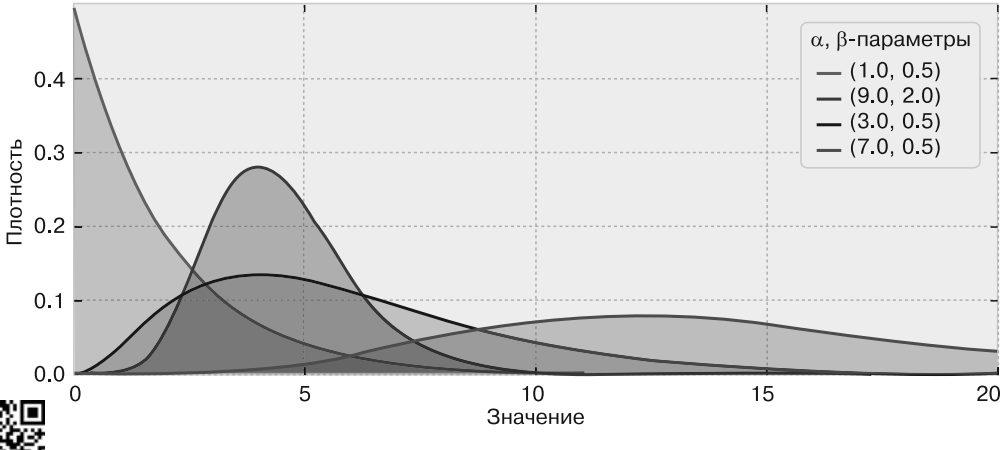

Рис. 6.2. Гамма-распределение при различных значениях α и β

6.3.2. Распределение Уишарта

До сих пор мы имели дело только со случайными переменными-скалярами (и векторами. — *Примеч. пер.*). Конечно, возможны и случайные матрицы! В частности, распределение Уишарта представляет собой распределение над пространством

положительно полуопределенных матриц. Чем оно удобно? Ковариационные матрицы положительно определены, следовательно, распределение Уишарта подходит в качестве априорного распределения для ковариационных матриц. Визуализировать распределение матриц невозможно, так что построим график некоторых реализаций из распределений Уишарта размерностью 4×4 (верхняя строка) и 15×15 (нижняя строка) (рис. 6.3).

```
Import pymc as pm

n = 4
hyperparameter = np.eye(n)
for i in range(5):
    ax = plt.subplot(2, 5, i+1)
    plt.imshow(pm.rwishart(n+1, hyperparameter), interpolation="none",
               cmap=plt.cm.hot)
    ax.axis("off")

n = 15
hyperparameter = 10*np.eye(n)
for i in range(5, 10):
    ax = plt.subplot(2, 5, i+1)
    plt.imshow(pm.rwishart(n+1, hyperparameter), interpolation="none",
               cmap=plt.cm.hot)
    ax.axis("off")

plt.suptitle(u"Случайные матрицы из распределения Уишарта");
```

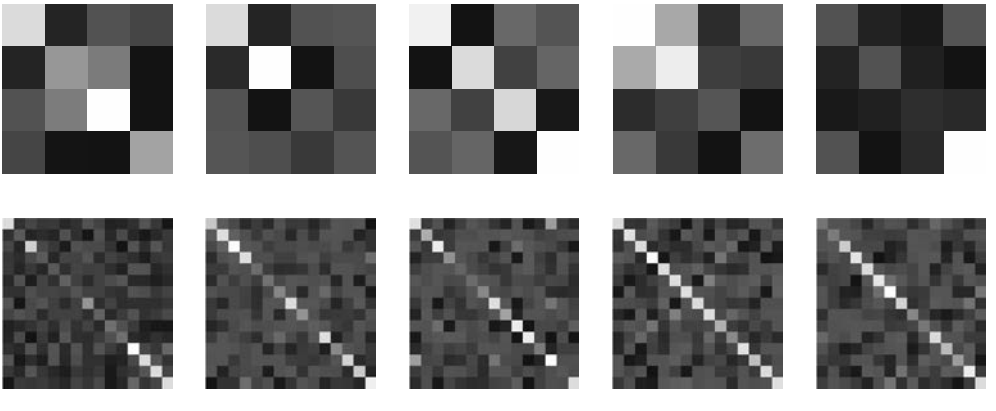

Рис. 6.3. Случайные матрицы 4×4 *(верхняя строка)* и 15×15 *(нижняя строка)* из распределения Уишарта

Обратите внимание на симметричность этих матриц, отражающую симметричность ковариационных матриц. Работать с распределением Уишарта непросто, но мы воспользуемся им в одном из дальнейших примеров.

6.3.3. Бета-распределение

Наверное, вы замечали слово `beta` в коде ранее в данной книге. Зачастую при этом реализовывалось **бета-распределение**. Бета-распределение оказывается весьма полезным в байесовской статистике. Случайная переменная распределена в соответствии с бета-распределением с параметрами (α, β), если ее функция плотности имеет вид:

$$f_X(x \mid \alpha, \beta) = \frac{x^{(\alpha-1)}(1-x)^{(\beta-1)}}{B(\alpha, \beta)},$$

где B — бета-функция (отсюда и название). Бета-распределение определяет случайные переменные со значениями от 0 до 1, благодаря чему с его помощью часто моделируют вероятности и процентные соотношения. Значения α и β положительные, они обеспечивают гибкость формы распределения. На рис. 6.4 приведены графики нескольких распределений при различных α и β:

```
figsize(12.5, 5)

params = [(2,5), (1,1), (0.5, 0.5), (5, 5), (20, 4), (5, 1)]

x = np.linspace(0.01, .99, 100)
beta = stats.beta
for a, b in params:
    y = beta.pdf(x, a, b)
    lines = plt.plot(x, y, label="(%.1f,%.1f)"%(a,b), lw = 3)
    plt.fill_between(x, 0, y, alpha=0.2, color=lines[0].get_color())
    plt.autoscale(tight=True)

plt.ylim(0)
plt.legend(loc='upper left', title=u"(a,b)-параметры")
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.title(ur"Бета-распределение при различных значениях $\alpha$ и $\beta$");
```

Я хотел бы обратить ваше внимание на то, что на предыдущем графике можно увидеть плоское распределение, которое получается при параметрах (1, 1). Это равномерное распределение. Следовательно, бета-распределение обобщает равномерное распределение, и мы этим еще не раз воспользуемся.

Существует любопытная взаимосвязь между бета-распределением и биномиальным распределением. Допустим, нас интересует какое-либо неизвестное соотношение или вероятность p. Зададим для p априорное распределение $B(\alpha, \beta)$. У нас есть наблюдаемые данные, полученные в результате какого-то биномиального процесса, скажем, $X \sim \text{Binomial}(N, p)$, причем параметр p неизвестен. Апостериорное распределение — опять же *бета-распределение*, то есть $p|X \sim \text{Beta}(\alpha + X, \beta + N - X)$. Кратко можно сформулировать это так: априорное бета-распределение с биномиаль-

ными наблюдениями приводит к апостериорному бета-распределению. Это очень удобное свойство как в вычислительном, так и эвристическом смысле.

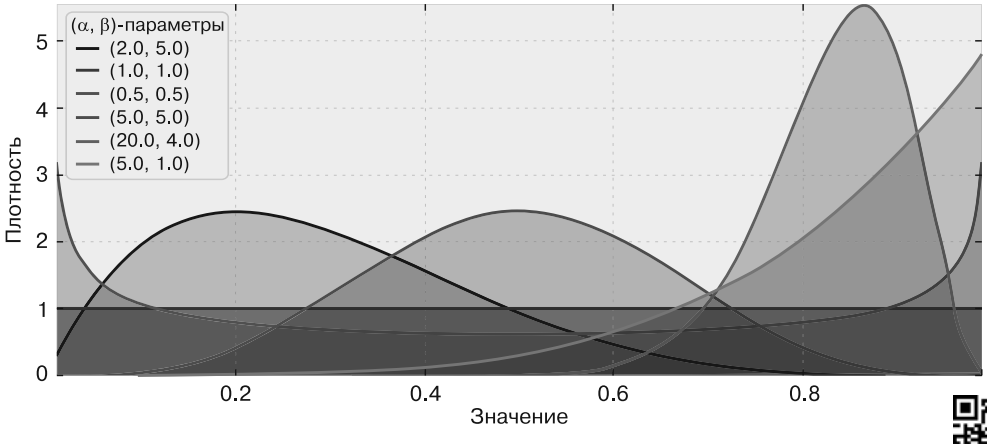

Рис. 6.4. Бета-распределение при различных значениях α и β

Конкретизируем немного: если начать с априорного распределения Beta(1, 1) для p (равномерное распределение) с наблюдаемыми данными $X \sim \text{Binomial}(N, p)$, то апостериорное распределение будет следующим: $\text{Beta}(1 + X, 1 + N - X)$. Например, при наблюдении $X = 10$ благоприятных исходов в $N = 25$ испытаниях апостериорное распределение будет иметь вид: $\text{Beta}(1 + 10, 1 + 25 - 10) = \text{Beta}(11, 16)$.

6.4. Пример: байесовские многорукие бандиты

Пусть дано десять игровых автоматов (образно называемых *многорукими бандитами*). У каждого из них своя вероятность выдачи приза (предположим пока, что призы одинаковы, различаются только вероятности). Часть бандитов очень «щедры», остальные — нет. Конечно, вероятности нам неизвестны. Наша задача — разработать стратегию максимизации выигрышей, выбирая по одному бандиту за раз.

Конечно, если бы мы знали, у какого бандита вероятность самая большая, то для максимизации выигрышей достаточно было бы всегда выбирать его. Так что нашу задачу можно переформулировать следующим образом: «Найти оптимальный бандит как можно быстрее».

Эту задачу усложняет стохастическая природа бандитов. Даже на не лучшем бандите совершенно случайно выигрыши могут оказаться большими, вследствие чего мы можем решить, что этот бандит — самый выгодный. Аналогично лучший бандит может совершенно случайно дать маленький выигрыш. Продолжать в подобном случае проверять плохие автоматы или сдаться?

Более сложная задача: если мы нашли автомат с *довольно неплохими показателями*, продолжать игру на нем или пробовать другие автоматы в надежде найти лучший? Это дилемма выбора — *исследовать или использовать*.

6.4.1. Приложения

Задача о многоруких бандитах на первый взгляд кажется совершенно искусственной и чисто математической, но лишь до того момента, как мы познакомимся с некоторыми ее приложениями.

- *Отображение рекламы в Интернете*. У компании есть набор рекламных объявлений, которые можно показать посетителям сайта, но непонятно, какой рекламной стратегии следовать для максимизации продаж. Это напоминает A/B-тестирование, но с дополнительным преимуществом: минимизацией неудачных групп естественным образом.
- *Экология*. Количество энергии, которое может потратить животное, ограниченно, а уверенности в вознаграждении при некоторых видах поведения нет. Как же животные максимизируют свою приспособляемость?
- *Финансы*. Доходность от какого биржевого опциона будет максимальной в случае меняющегося со временем профиля доходности?
- *Клинические испытания*. Исследователь хочет найти наилучший вариант лечения из множества возможных с минимизацией потерь.

Оказывается, что найти оптимальное решение невероятно сложно, а нахождение комплексного решения требует десятков лет. Однако существует также множество довольно неплохих, приближенных к оптимальному решений. В этой главе я хочу обсудить одно из них, хорошо масштабируемое и легко модифицируемое. Оно известно под названием *«байесовские бандиты»* (или «стратегия байесовских бандитов») [3].

6.4.2. Предлагаемое решение

Алгоритм начинает работу, не зная ничего, и постепенно получает данные в результате испытаний системы. По мере получения данных и результатов алгоритм выясняет наилучшие и наихудшие варианты поведения (в данном случае определяет, какой из автоматов наилучший). С учетом этого можно добавить еще одно приложение задачи о многоруких бандитах.

- *Психология*. Как наказание и вознаграждение влияют на наше поведение? Как люди учатся?

Байесовское решение начинается с задания априорных распределений вероятностей выигрыша для каждого из бандитов. В кратком описании мы предположили

полное отсутствие информации об этих вероятностях, так что наиболее логичным априорным распределением будет плоское распределение на отрезке от 0 до 1. Алгоритм выглядит следующим образом.

1. Извлечь случайную выборку X_b из априорного распределения бандита b для каждого b.
2. Выбрать бандит с максимальной выборкой, то есть выбрать $B = \text{argmax } X_b$.
3. Посмотреть на результаты испытания для бандита B и обновить для него априорное распределение.
4. Вернуться к шагу 1.

Вот и все. С вычислительной точки зрения этот алгоритм включает выполнение выборки из N распределений. Поскольку исходные априорные распределения представляют собой Beta($\alpha = 1, \beta = 1$), то есть равномерные распределения, а наблюдаемый результат X (выигрыш или потеря денег, закодированные как 1 и 0 соответственно) биномиален, то апостериорное распределение имеет вид Beta($\alpha = 1 + X$, $\beta = 1 + 1 - X$).

Для ответа на вышеприведенный вопрос алгоритм предполагает, что нужно не отбрасывать неудачные бандиты, а просто выбирать их все реже и реже, по мере получения информации о том, что существуют лучшие. Это логично, поскольку всегда существует ненулевая вероятность того, что плохой бандит окажется бандитом B с наибольшим значением выборки, но эта вероятность уменьшается с каждой итерацией (рис. 6.5).

Реализуем далее алгоритм байесовских бандитов с помощью двух классов: `Bandits`, который описывает игровые автоматы, и `BayesianStrategy`, который реализует вышеописанную стратегию обучения.

```
from pymc import rbeta

class Bandits(object):
    """
    Этот класс олицетворяет N игровых автоматов.

    параметры:
        p_array: NumPy-массив формы (N,) вероятностей >0, <1.

    методы:
        pull(i): возвращает результат, 0 or 1, испытания для i-го бандита.
    """
    def __init__(self, p_array):
        self.p = p_array
        self.optimal = np.argmax(p_array)

    def pull(self, i):
        # i — номер бандита.
```

```
        # Возвращает True в случае выигрыша, False в противном случае.
        return np.random.rand() < self.p[i]

    def __len__(self):
        return len(self.p)

class BayesianStrategy(object):
    """
    Реализует стратегию обучения для решения задачи о многоруких бандитах.

    параметры:
        bandits: класс Bandit с методом .pull

    методы:
        sample_bandits(n): выборка и обучение на n испытаниях.

    атрибуты:
        N: совокупное число выборок
        choices: история выбранных бандитов в виде массива формы (N,)
        bb_score: история показателей бандитов в виде массива формы (N,)
    """

    def __init__(self, bandits):

        self.bandits = bandits
        n_bandits = len(self.bandits)
        self.wins = np.zeros(n_bandits)
        self.trials = np.zeros(n_bandits)
        self.N = 0
        self.choices = []
        self.bb_score = []

    def sample_bandits(self, n=1):

        bb_score = np.zeros(n)
        choices = np.zeros(n)

        for k in range(n):
            # Выполняем выборку из априорного распределения бандита
            # и берем выборку с наибольшим значением
            choice = np.argmax(rbeta(1 + self.wins, 1 + self.trials
            - self.wins))
            # испытание заданного бандита
            result = self.bandits.pull(choice)

            # обновление априорного распределения и показателей бандита
            self.wins[choice] += result
            self.trials[choice] += 1
            bb_score[k] = result
            self.N += 1
```

```
            choices[k] = choice

        self.bb_score = np.r_[self.bb_score, bb_score]
        self.choices = np.r_[self.choices, choices]
        return
```

На рис. 6.5 мы визуализируем ход выполнения алгоритма байесовских бандитов.

```
figsize(11.0, 10)

beta = stats.beta
x = np.linspace(0.001,.999,200)

def plot_priors(bayesian_strategy, prob, lw=3, alpha=0.2, plt_vlines=True):
    # Строим график функции
    wins = bayesian_strategy.wins
    trials = bayesian_strategy.trials
    for i in range(prob.shape[0]):
        y = beta(1 + wins[i], 1 + trials[i] - wins[i])
        p = plt.plot(x, y.pdf(x), lw=lw)
        c = p[0].get_markeredgecolor()
        plt.fill_between(x,y.pdf(x),0 ,color=c, alpha=alpha,
                 label=u"исходная вероятность: %.2f"%prob[i])
        if plt_vlines:
            plt.vlines(prob[i], 0, y.pdf(prob[i]),
                 colors=c, linestyles="--", lw=2)
        plt.autoscale(tight="True")
        plt.title(u"Апостериорные распределения после %d испытаний"\
             % bayesian_strategy.N + u"я"*(bayesian_strategy.N == 1)
             + u"й" * (bayesian_strategy.N > 1))
        plt.autoscale(tight=True)
    return

    hidden_prob = np.array([0.85, 0.60, 0.75])
    bandits = Bandits(hidden_prob)
    bayesian_strat = BayesianStrategy(bandits)

    draw_samples = [1, 1, 3, 10, 10, 25, 50, 100, 200, 600]

    for j,i in enumerate(draw_samples):
        plt.subplot(5, 2, j+1)
        bayesian_strat.sample_bandits(i)
        plot_priors(bayesian_strat, hidden_prob)
        plt.autoscale(tight=True)
    plt.xlabel(u'Значение')
    plt.ylabel(u'Плотность')
    plt.title(u"Апостериорные распределения, полученные\nв результате
              наших выводов относительно\nкаждого из бандитов после
              различного\nколичества испытаний")
    plt.tight_layout()
```

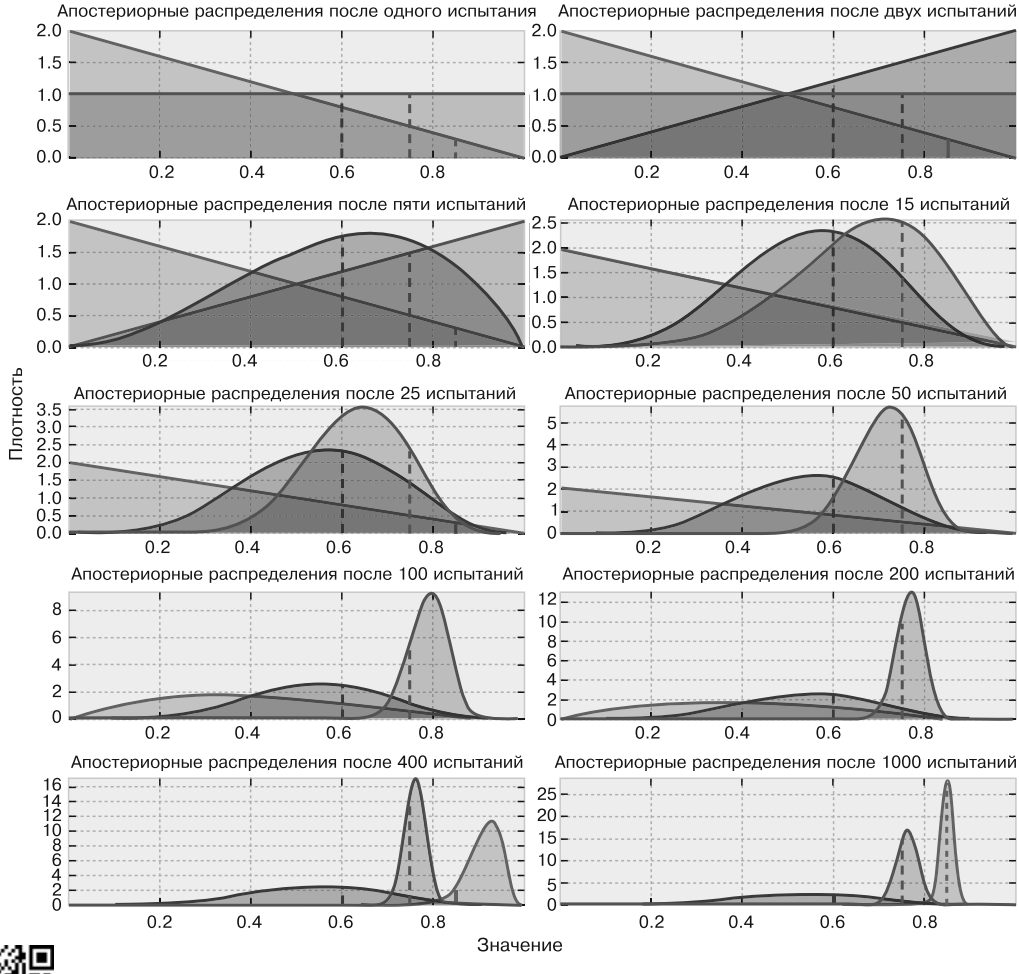

Рис. 6.5. Апостериорные распределения, полученные в результате наших выводов относительно каждого из бандитов после различного количества испытаний

Отмечу, что нас не особо волнует точность выводов относительно скрытых вероятностей — в данной задаче нас больше интересует выбор оптимального бандита (или, точнее говоря, повышение нашей уверенности в выборе оптимального бандита). Поэтому распределение «красного» бандита — очень широкое (отражая незнание нами значения скрытой вероятности), но мы в достаточной степени уверены, что он не лучший, так что алгоритм его пропускает.

Из рис. 6.5 видно, что после 1000 испытаний лидирует «синий» бандит, так что мы почти всегда будем выбирать его. И это хорошо, ведь он, несомненно, лучший.

Отклонение наблюдаемого соотношения от максимальной вероятности — мера эффективности. Например, в лучшем случае при длительном выполнении алгоритма мы достигнем соотношения «вознаграждение/испытания», соответствующего максимальной вероятности для бандитов. Полученные в результате длительного выполнения алгоритма соотношения, меньшие, чем этот максимум, означают неэффективную работу (превышать этот максимум соотношения могут только вследствие случайности процесса, и в конце концов они должны снизиться до его уровня или ниже).

6.4.3. Мера качества

Нам требуется метрика, которая бы отражала успешность нашей работы. Напомню, что самый идеальный для нас случай — если мы всегда выбираем бандита с максимальной вероятностью выигрыша. Обозначим вероятность выигрыша этого оптимального для нас бандита $w_{\text{опт}}$. Показатели бандитов должны измеряться относительно показателей, которые мы получили бы при выборе с самого начала оптимального бандита. Это побуждает ввести понятие **совокупного сожаления** (total regret) стратегии, определяемого как разница между отдачей от выбора оптимальной стратегии в течение T итераций (выбора бандита с максимальной вероятностью выигрыша) и отдачей от выбора в течение T итераций другой стратегии. На формальном математическом языке оно определяется следующим образом:

$$R_T = \sum_{i=1}^{T} (w_{\text{опт}} - w_{B(i)}) =$$

$$= Tw^* - \sum_{i=1}^{T} w_{B(i)}.$$

В этом уравнении $w_{B(i)}$ — вероятность получения приза для выбранного бандита на i-й итерации. Равное 0 совокупное сожаление означает, что стратегия достигла наилучших показателей. Вероятно, это невозможно, поскольку поначалу выбор алгоритма часто будет неправильным. В идеальном случае график совокупного сожаления выравнивается по мере выяснения того, какой бандит лучший для нас (математически говоря, часто достигается равенство $w_{B(i)} = w_{\text{опт}}$).

На рис. 6.6 приведен график общего сожаления нашего имитационного моделирования, в том числе показатели еще нескольких стратегий.

1. Случайная стратегия (в коде ниже ей соответствует модуль `random_choice`). Бандит выбирается случайным образом. Если не удается превзойти эту стратегию, то лучше просто прекратить попытки.
2. Верхняя граница байесовского доверительного интервала (модуль `upper_credible_choice`). Выбирается бандит с максимальной верхней границей в его 95%-ной байесовской доверительной области базовой вероятности.

3. Алгоритм UCB-байес[1] (модуль `ucb_bayes`). Выбирается бандит с максимальным *количественным показателем*, где в качестве количественного показателя служит динамический квантиль апостериорного распределения (см. [2]).
4. Среднее значение апостериорного распределения. Выбирается бандит с максимальным средним значением апостериорного распределения. Именно так обычно поступают люди-игроки (без компьютера).
5. Максимальное отношение числа выигрышей к числу испытаний (модуль `max_mean`). Выбирается бандит с максимальным на текущий момент наблюдаемым отношением числа выигрышей к числу испытаний.

Код для этого примера находится в файле `other_strats.py`. Используя его, вы с легкостью можете реализовать и свои собственные стратегии.

```
figsize(12.5, 5)
from other_strats import upper_credible_choice, bayesian_bandit_choice,
     ucb_bayes, max_mean, random_choice

# Описание задачи
hidden_prob = np.array([0.15, 0.2, 0.1, 0.05])
bandits = Bandits(hidden_prob)

# Описание функции вычисления сожаления
def regret(probabilities, choices):
    w_opt = probabilities.max()
    return(w_opt - probabilities[choices.astype(int)]).cumsum()

# Создаем новые стратегии
strategies= [upper_credible_choice,
      bayesian_bandit_choice,
      ucb_bayes,
      max_mean,
      random_choice]
algos = []
for strat in strategies:
    algos.append(GeneralBanditStrat(bandits, strat))

# Обучение (10 000 выборок)
for strat in algos:
    strat.sample_bandits(10000)

# Вычисление сожалений и построение графика
for i,strat in enumerate(algos):
    _regret = regret(hidden_prob, strat.choices)
    plt.plot(_regret, label=strategies[i].__name__, lw = 3)
```

[1] От англ. upper confidence bounds — «верхние доверительные границы».

```
plt.title(u"Совокупное сожаление стратегии байесовских бандитов
          по сравнению со случайным гаданием")
plt.xlabel(u"Количество испытаний")
plt.ylabel(u"Сожаление после $n$ испытаний")
plt.legend(loc="upper left");
```

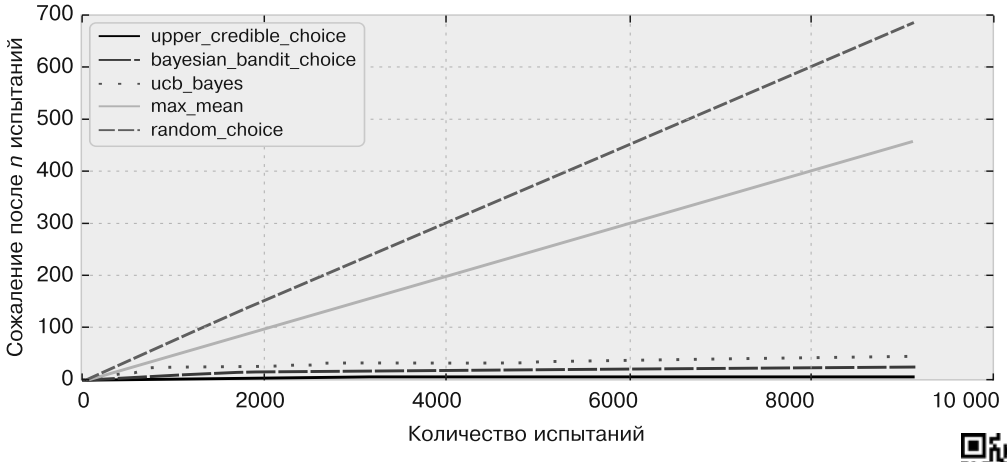

Рис. 6.6. Совокупное сожаление стратегии байесовских бандитов по сравнению со случайным гаданием

Как мы и хотели, уровень сожаления в стратегии байесовских бандитов и других стратегиях понижается, демонстрируя достижение оптимальных вариантов. Для исключения любого элемента случайности в предыдущем имитационном моделировании и повышения его «научности» следовало бы ориентироваться на **ожидаемое совокупное сожаление** — математическое ожидание совокупного сожаления по всем возможным сценариям, математически определяемое как:

$$R_T^- = E[R_T].$$

Можно показать, что ожидаемое совокупное сожаление любой не оптимальной стратегии ограничено снизу логарифмически. На формальном языке:

$$E[R_T] = \Omega(\log(T)).$$

Так, считается, что любая стратегия с логарифмически растущим сожалением решает задачу о многоруких бандитах [4].

С помощью закона больших чисел можно приближенно вычислить ожидаемое совокупное сожаление стратегии байесовских бандитов, многократно выполняя один и тот же эксперимент (скажем, 200 раз для надежности). Результаты показаны

на рис. 6.7. А чтобы нагляднее продемонстрировать различия стратегий, мы построим тот же график на логарифмической шкале (рис. 6.8).

```
# Этот код может работать очень долго, так что рекомендую его НЕ запускать.

trials = 200
expected_total_regret = np.zeros((1000, 3))

for i_strat, strat in enumerate(strategies[:-2]):
    for i in range(trials):
        general_strat = GeneralBanditStrat(bandits, strat)
        general_strat.sample_bandits(1000)
        _regret = regret(hidden_prob, general_strat.choices)
        expected_total_regret[:,i_strat] += _regret

    plt.plot(expected_total_regret[:,i_strat]/trials, lw =3,
         label = strat.__name__)

plt.title(u"Ожидаемое совокупное сожаление различных стратегий для задачи
          о многоруких бандитах")
plt.xlabel(u"Число испытаний")
plt.ylabel(u"Ожидаемое совокупное сожаление после $n$ испытаний")
plt.legend(loc="upper left");
plt.figure()
[pl1, pl2, pl3] = plt.plot(expected_total_regret[:, [0,1,2]], lw = 3)
plt.xscale("log")
plt.legend([pl1, pl2, pl3],
      ["upper_credible_choice", "bayesian_bandit_choice", "ucb_bayes"],
      loc="upper left")
plt.ylabel(u"Ожидаемое совокупное сожаление\n после $\log{n}$ испытаний")
plt.xlabel(u"Число испытаний, $n$")
plt.title(u"Ожидаемое совокупное сожаление различных стратегий\n\
          решения задачи о многоруких бандитах на логарифмической шкале");
```

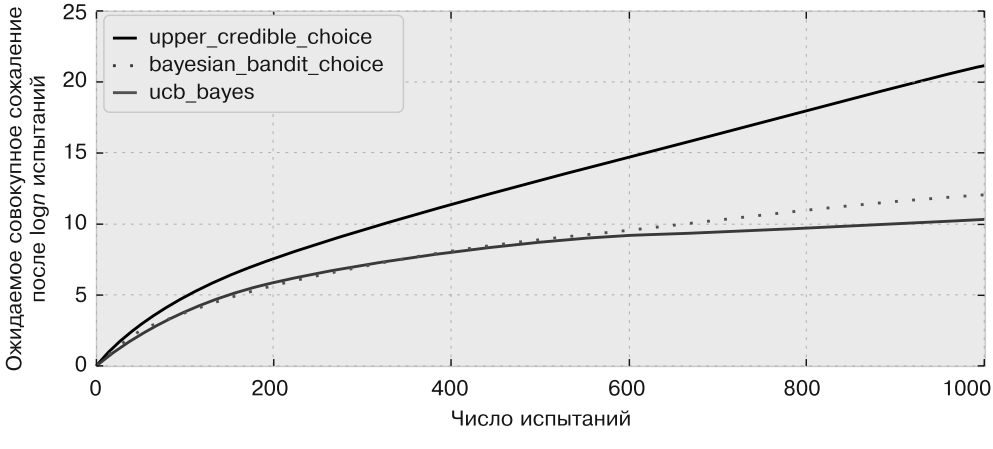

Рис. 6.7. Ожидаемое совокупное сожаление различных стратегий для задачи о многоруких бандитах

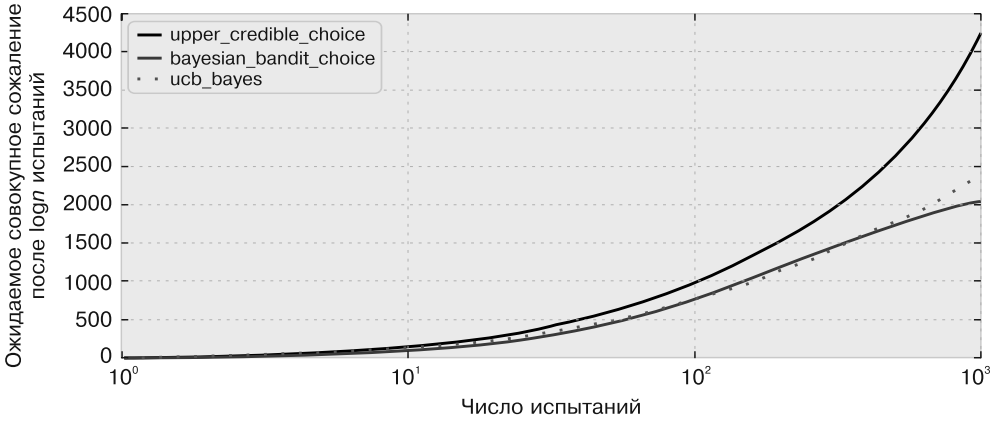

Рис. 6.8. Совокупное сожаление различных стратегий решения задачи о многоруких бандитах на логарифмической шкале

6.4.4. Обобщения алгоритма

Благодаря простоте алгоритма байесовских бандитов его обобщение не представляет сложностей. Вот некоторые из вариантов.

- Если вас интересует *минимальная* вероятность (например, когда выигрыш призов нежелателен), просто выберите бандита $B = \text{argmin } X_b$, в остальном алгоритм не меняется.
- Добавляем в алгоритм понятие темпов обучения. Допустим, что исходные условия со временем могут меняться. Формально обычный алгоритм байесовских бандитов должен самокорректироваться (потрясающе!) в случае обнаружения плохих показателей бандита, который он считал лучшим. Чтобы побудить алгоритм быстрее учитывать изменения окружающих условий, можно просто добавить коэффициент *темпа* корректировки алгоритма.

  ```
  self.wins[ choice ] = rate*self.wins[ choice ] + result
  self.trials[ choice ] = rate*self.trials[ choice ] + 1
  ```

  При `rate` < 1 алгоритм быстрее «забывает» предыдущие выигрыши. И напротив, если задать `rate` > 1, то алгоритм будет вести себя более «рискованно», чаще делая ставку на ранее выигрывавших бандитов и демонстрируя меньшую чувствительность к изменению окружающих условий.
- Задействуем иерархические алгоритмы. Можно использовать алгоритм байесовских бандитов как надстройку к другим алгоритмам решения задачи о бандитах. Пусть у нас есть N моделей для байесовских бандитов, различающихся чем-либо в поведении (например, различными значениями параметра `rate`, а значит, различной чувствительностью к изменению окружающих условий).

Поверх этих N моделей работает другой обучающийся алгоритм для выбора байесовских бандитов нижнего уровня. Выбранные таким образом модели байесовских бандитов неявно выбирают автомат для испытания. А затем происходит корректировка моделей байесовских бандитов верхнего уровня на основе того, выиграл байесовский бандит нижнего уровня или нет.

- Вполне очевидным ходом представляется обобщение модели вознаграждений (вознаграждение для бандита A обозначается y_a) до случайных переменных с распределением f_{γ_a} (γ). В более общем виде эту задачу можно переформулировать так: «Найти бандит с максимальным математическим ожиданием», поскольку оптимально играть именно на бандите с максимальным математическим ожиданием. В предыдущем случае f_{γ_b} была бернуллиевой случайной переменной с вероятностью p_a, следовательно, математическое ожидание бандита будет равно p_a, именно поэтому складывается впечатление, что мы стремимся максимизировать вероятность выигрыша. Если же приз не бернуллиева случайная переменная, причем он неотрицателен (чего можно достичь за счет сдвига распределения при условии, что нам известно f), то алгоритм будет работать так же, как и ранее.

Для каждой итерации нужно сделать следующее.

1. Извлечь случайную переменную X_b из априорного распределения бандита b для всех b.
2. Выбрать бандит с максимальной выборкой, то есть выбрать $B = \text{argmax } X_b$.
3. Изучить результат $R \sim f_{\gamma_b}$ испытания бандита B и обновить для него априорное распределение.
4. Вернуться к шагу 1.

Основная проблема возникает на этапе 1. В случае априорных бета-распределений и распределенных по закону Бернулли наблюдений апостериорное распределение будет представлять собой бета-распределение, выборка из которого не представляет сложностей. Но пока при произвольных распределениях f апостериорное распределение также будет нетривиальным. Выборка из него может оказаться непростой задачей.

Сейчас наблюдается интерес к обобщению алгоритма байесовских бандитов на системы комментариев. Напомню, что в главе 4 мы разработали алгоритм ранжирования на основе нижней байесовской границы отношения числа голосов «за» к суммарному числу всех голосов. Одна из проблем с этим подходом заключалась в его тенденции присваивать самый высокий рейтинг более старым комментариям, поскольку у них, естественно, больше голосов (и, следовательно, нижняя граница ближе к фактическому соотношению). В результате создается петля положительной обратной связи, в которой более старые комментарии получают больше голосов,

а потому отображаются чаще, вследствие чего получают больше голосов и т. д. В итоге новые (возможно, лучшие) комментарии опускаются в рейтинге вниз.

Дж. Нойфельд (J. Neufeld) предложил лишенную этого недостатка систему на основе алгоритма байесовских бандитов. Он предложил рассматривать каждый комментарий как аналог бандита с числом испытаний, равным числу голосов, и числом голосов «за» в качестве вознаграждения, в результате чего получается апостериорное распределение Beta(1 + U, 1 + D). При входе посетителя на страницу производятся выборки из всех бандитов/комментариев, но вместо отображения комментария с максимальным значением выборки комментарии ранжируются согласно рейтингу соответствующих выборок. Из блога Дж. Нойфельда [5]:

> «Получившийся в итоге алгоритм ранжирования довольно прост: при каждой загрузке страницы с комментариями из распределения Beta(1 + U, 1 + D) осуществляется выборка показателей для каждого из комментариев и комментарии располагаются в убывающем порядке в соответствии с этими показателями... Уникальное преимущество такого внесения элемента случайности состоит в шансе даже для комментариев без голосов[1] быть замеченными в ветках обсуждения из более чем 5000 комментариев (чего сейчас не происходит), но в то же время пользователи вряд ли будут завалены оценками этих новых комментариев».

Просто из спортивного интереса посмотрим, как алгоритм байесовских бандитов обучается на 35 различных байесовских бандитах (рис. 6.9).

```
figsize(12.0, 8)
beta = stats.beta
hidden_prob = beta.rvs(1,13, size=35)

print hidden_prob
bandits = Bandits(hidden_prob)
bayesian_strat = BayesianStrategy(bandits)

for j,i in enumerate([100, 200, 500, 1300]):
    plt.subplot(2, 2, j+1)
    bayesian_strat.sample_bandits(i)
    plot_priors(bayesian_strat, hidden_prob, lw = 2, alpha = 0.0, plt_vlines=False)
    plt.xlim(0, 0.5)
```

```
[Output]:

[ 0.2411 0.0115 0.0369 0.0279 0.0834 0.0302 0.0073 0.0315 0.0646
  0.0602 0.1448 0.0393 0.0185 0.1107 0.0841 0.3154 0.0139 0.0526
  0.0274 0.0885 0.0148 0.0348 0.0258 0.0119 0.1877 0.0495 0.236
  0.0768 0.0662 0.0016 0.0675 0.027  0.015  0.0531 0.0384]
```

[1] U = 0, D = 0, где U — число голосов «за», а D — число голосов «против».

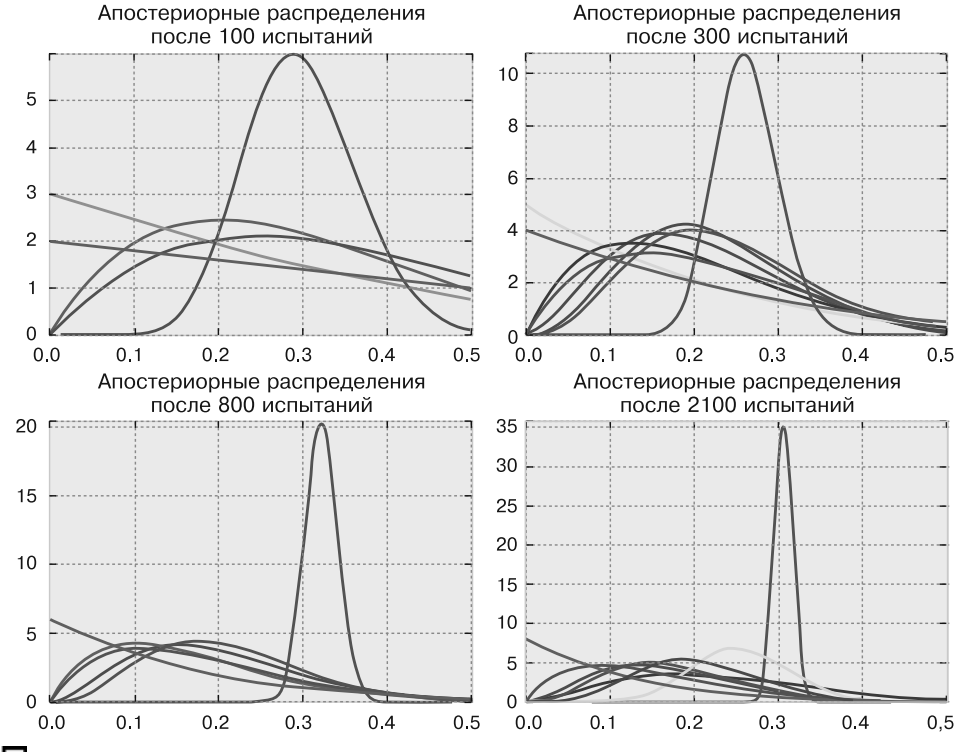

Рис. 6.9. Эволюция стратегии байесовских бандитов при обучении на 35 различных бандитах

6.5. Сбор информации для априорных распределений у специалистов по предметной области

Знания о предметной области включаются в математическую модель с помощью субъективного априорного распределения. Учет знаний о предметной области полезен по многим причинам.

- Они повышают сходимость MCMC. Например, если мы знаем, что неизвестный параметр строго положителен, то можем ограничить свое внимание областью положительных чисел и сэкономить время, которое в противном случае было бы потрачено на отрицательные числа.
- Они лучше выражают неопределенность. См. задачу «Справедливая цена» из главы 5.

- Они повышают точность вывода. Повышение весов априорных значений вблизи фактического значения неизвестной величины сужает область вывода (благодаря сужению апостериорного распределения возле него).

Конечно, специалисты, использующие байесовские методы на практике, не могут быть экспертами во всех сферах, так что для создания априорных распределений приходится обращаться к экспертам в данной предметной области. Однако следует с осторожностью подходить к выяснению этих априорных распределений. Есть нюансы, которые стоит учесть.

- Я избегал бы упоминания различных бета- и гамма-распределений в кругу специалистов, не знакомых с байесовскими методами. Более того, неспециалистов в статистике может привести в замешательство тот факт, что значение непрерывной функции распределения вероятности может превышать 1.
- Отдельные специалисты часто пренебрегают редкими «хвостовыми» событиями и придают большой вес области вокруг среднего значения распределения.
- По отдельности специалисты практически всегда недооценивают неопределенность в своих гипотезах.

Выяснение априорных распределений у экспертов-гуманитариев — особенно сложная задача. Существует намного более простой вариант, не требующий знакомства с понятиями распределения вероятности, априорных распределений и т. д., которые могут испугать такого эксперта.

6.5.1. Метод рулетки испытаний

Метод рулетки испытаний [6] основывается на формировании априорного распределения с помощью распределения фишек (таких как в казино) по возможным (с точки зрения эксперта) исходам. Эксперту дают N фишек (пусть $N = 20$) и просят его разместить их на заранее размеченной координатной сетке, где ячейки соответствуют интервалам. Каждый столбец при этом отражает степень уверенности эксперта в попадании исхода в конкретную ячейку. Каждая фишка соответствует повышению вероятности попадания исхода в этот интервал на $\frac{1}{N} = 0,05$. Например, в [7] рассматривается ситуация, когда студенту нужно предсказать будущую оценку на экзамене. На рис. 6.10 показана сетка, заполненная собранной информацией о субъективном распределении вероятностей. Горизонтальная ось координатной сетки отражает возможные ячейки (интервалы), предлагаемые студенту для выбора. Числа, соответствующие верхней строке, означают число фишек в ячейке. Заполненная (20 фишками) сетка демонстрирует, что студент считает вероятность оценки между 50 и 59,9 равной 30 %[1].

[1] По 100-балльной (болонской) шкале 50–60 баллов — это примерно тройка.

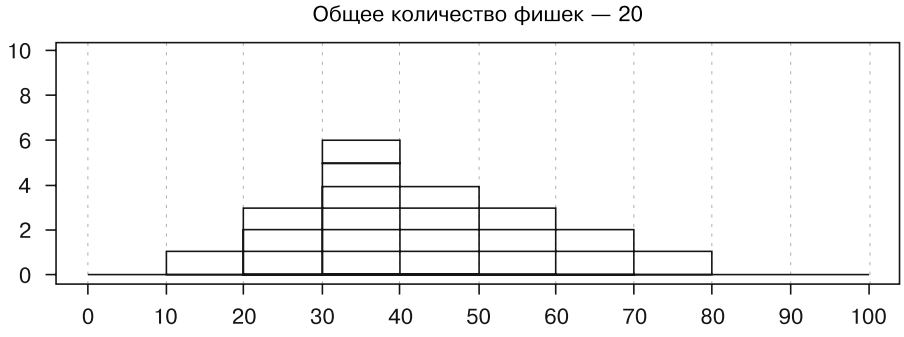

Рис. 6.10. Метод рулетки испытаний для сбора информации у экспертов об априорных распределениях (рисунок из [8])

На основе этой информации можно сформировать распределение, отражающее выбор эксперта. Существует несколько доводов в пользу этой методики.

- На многие вопросы о форме субъективного для эксперта распределения вероятностей можно ответить и без того, чтобы задавать ему множество вопросов. Специалист-статистик может просто подсчитать плотность вероятности ниже заданной точки или между двумя точками.
- Во время создания априорного распределения эксперт может переставлять фишки, если исходное их расположение перестало его устраивать. Таким образом эксперт может быть уверен в представляемом итоговом результате.
- Эксперт вынужден предоставить логически непротиворечивый набор вероятностей. Если все фишки использованы, то сумма вероятностей должна равняться 1.
- Похоже, что результаты при использовании визуальных методов более точны, особенно для участников с небольшим опытом в области статистики.

6.5.2. Пример: биржевая прибыль

Биржевые брокеры, обратите внимание: вы все делаете неправильно. Аналитик, выбирая, какие акции покупать, зачастую ориентируется на *доход в день* (daily return). Если S_t — цена акции в день t, то доход от нее в день t равен:

$$r_t = \frac{S_t - S_{t-1}}{S_{t-1}}.$$

Ожидаемый доход в день от акции обозначается $\mu = E[r_t]$. Разумеется, всех интересуют акции с высокой ожидаемой доходностью. К сожалению, данные о биржевых доходах настолько «зашумлены», что оценить этот параметр очень непро-

сто. Более того, со временем он может меняться (взгляните, например, на взлеты и падения акций AAPL — Apple Inc.), поэтому использовать неразумно большой набор данных.

Ранее ожидаемый доход оценивался с помощью выборочного среднего. Это не лучшая идея. Как уже упоминалось, шансы на то, что выборочное среднее маленького набора данных не подходит для наших целей, огромны (см. подробности в главе 4). Следовательно, здесь отлично подойдет байесовский вывод, который позволит видеть вместе с возможными значениями и степень неопределенности.

В этом примере мы изучим показатели дохода в день акций AAPL, GOOG (Google), TSLA (Tesla Motors) и AMZN (Amazon.com Inc.). Доход в день этих видов акций проиллюстрирован на рис. 6.12 и 6.13. Прежде чем приступить к изучению данных, представьте себе, что мы задаем нашему фондовому менеджеру (эксперту в сфере финансов, см. [9]) вопрос: «Каким, по твоему мнению, будет профиль доходности каждой из этих компаний?» Наш биржевой брокер, не используя никаких терминов вроде «нормальное распределение», «априорное распределение» или «дисперсия», создает четыре распределения с помощью метода рулетки испытаний, подробно изложенного в предыдущем подразделе. Допустим, они достаточно похожи на нормальные распределения. На рис. 6.11 показан возможный вариант априорных распределений, полученных от биржевого брокера.

```
figsize(11.0, 5)
colors = ["#348ABD", "#A60628", "#7A68A6", "#467821"]

normal = stats.norm
x = np.linspace(-0.15, 0.15, 100)

expert_prior_params = {"AAPL":(0.05, 0.03),
    "GOOG":(-0.03, 0.04),
    "TSLA": (-0.02, 0.01),
    "AMZN": (0.03, 0.02),
    }

for i, (name, params) in enumerate(expert_prior_params.iteritems()):
    plt.subplot(2,2,i)
    y = normal.pdf( x, params[0], scale=params[1] )
    plt.fill_between(x, 0, y, color=colors[i], linewidth=2,
             edgecolor=colors[i], alpha=0.6)
    plt.title(u"Априорное распределение " + name)
    plt.vlines(0, 0, y.max(), "k","--", linewidth=0.5)
    plt.xlim(-0.15, 0.15)
    plt.tight_layout()
```

Обратите внимание, что это субъективные априорные распределения: у эксперта есть собственное мнение относительно биржевой доходности каждой из этих компаний, которое он и выражает в распределениях. Впрочем, он не выдает желаемое за действительное, а просто включает в модель свои знания предметной области.

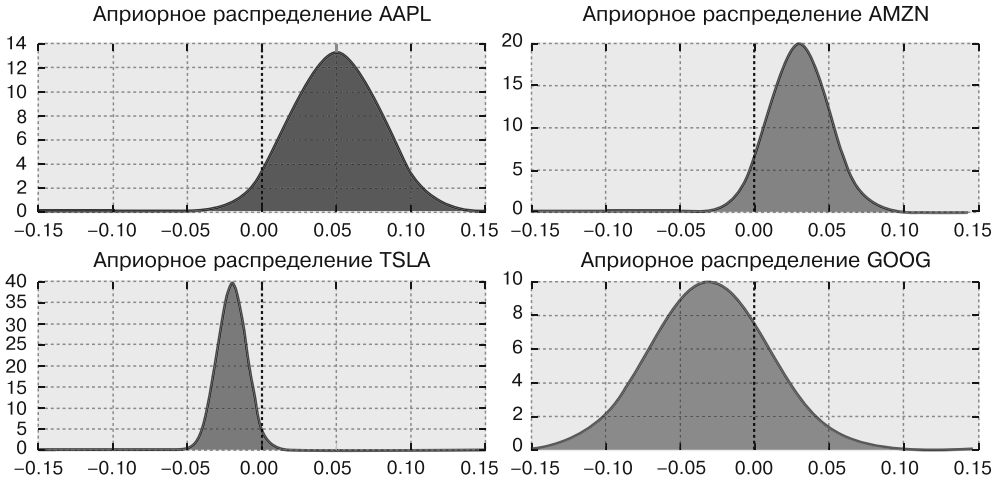

Рис. 6.11. Априорные распределения доходов от различных котирующихся на бирже акций

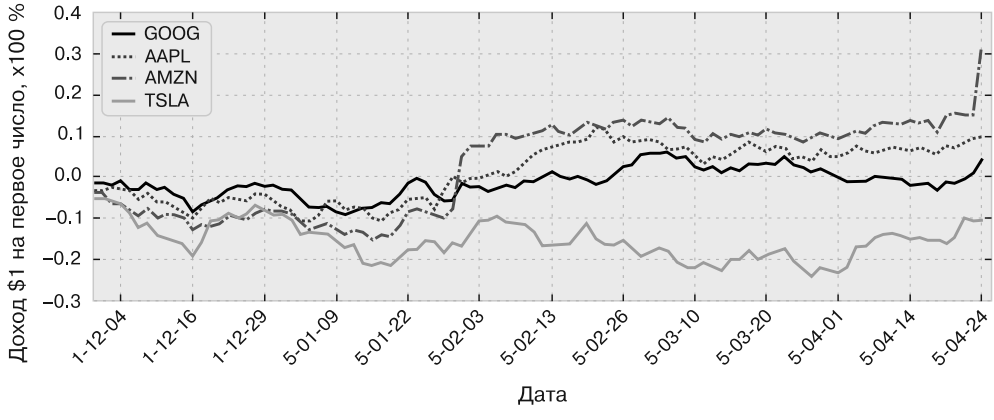

Рис. 6.12. Представление цен на акции в пространстве доходов

Для более качественного моделирования этих доходов имеет смысл изучить матрицу ковариации доходов. Например, неразумно инвестировать в два вида сильно коррелированных акций, поскольку их курсы, вероятно, будут падать синхронно (именно поэтому фондовые менеджеры рекомендуют стратегии диверсификации). Для этой цели мы воспользуемся описанным в подразделе 6.3.2 распределением Уишарта.

```
import pymc as pm

n_observations = 100 # Ограничиваемся последними 100 днями.
prior_mu = np.array([x[0] for x in expert_prior_params.values()])
```

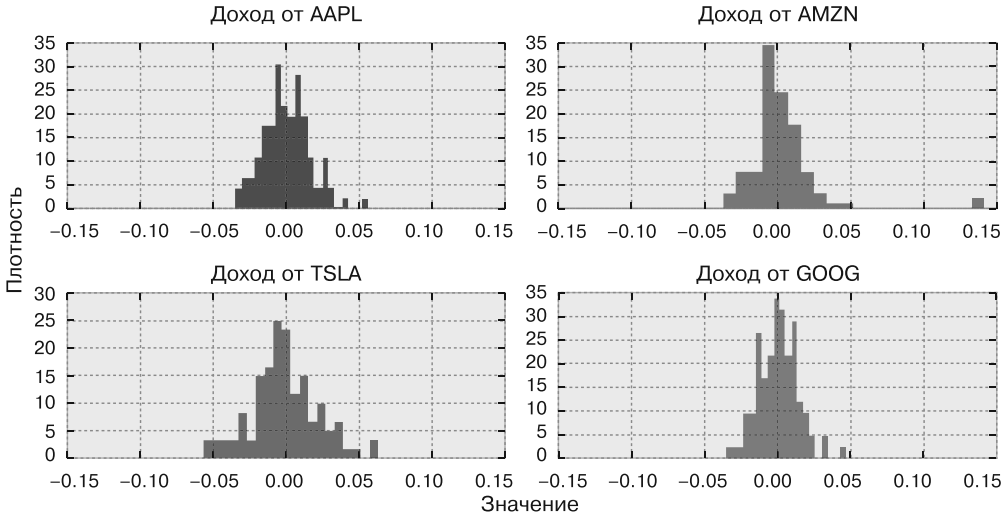

Рис. 6.13. Гистограммы доходов в день от акций

```
prior_std = np.array([x[1] for x in expert_prior_params.values()])

inv_cov_matrix = pm.Wishart("inv_cov_matrix", n_observations, np.diag
                 (prior_std**2))
mu = pm.Normal("returns", prior_mu, 1, size=4)
```

Извлечем[1] теперь необходимые данные по этим видам акций.

```
import datetime
import ystockquote as ysq

stocks = ["AAPL", "GOOG", "TSLA", "AMZN"]

enddate = datetime.datetime.now().strftime("%Y-%m-%d") # today's date
startdate = "2012-09-01"

stock_closes = {}
stock_returns = {}
CLOSE = 6

for stock in stocks:
```

[1] К сожалению, судя по всему, компания Yahoo закрыла доступ к соответствующему (неофициальному) API, поэтому приведенный ниже код в настоящее время неработоспособен. В качестве нетривиального упражнения для читателей можем предложить переписать прилагаемый к книге файл ystockquote.py для использования финансового API Google либо какого-либо аналогичного.

```
    x = np.array(ysq.get_historical_prices(stock, startdate, enddate))
    stock_closes[stock] = x[1:,CLOSE].astype(float)

# Формируем данные о доходах

for stock in stocks:
    _previous_day = np.roll(stock_closes[stock], -1)
    stock_returns[stock] = ((stock_closes[stock] - _previous_day)/
                            _previous_day)[:n_observations]

dates = map(lambda x: datetime.datetime.strptime(x, "%Y-%m-%d"),
            x[1:n_observations+1,0])

figsize(12.5, 4)

for _stock, _returns in stock_returns.iteritems():
    p = plt.plot((1+_returns)[::-1].cumprod()-1, '-o', label="%s"%_stock,
        markersize=4, markeredgecolor="none" )

plt.xticks( np.arange(100)[::-8],
       map(lambda x: datetime.datetime.strftime(x, "%Y-%m-%d"), dates[::8]),
       rotation=60);

plt.legend(loc="upper left")
plt.title(u"Представление цен на акции в пространстве доходов")
plt.xlabel(u"Дата")
plt.ylabel(u"Доход в $1 на первое число, x100%");
figsize(11.0, 5)
returns = np.zeros((n_observations,4))

for i, (_stock,_returns) in enumerate(stock_returns.iteritems()):
    returns[:,i] = _returns
    plt.subplot(2,2,i)
    plt.hist( _returns, bins=20,
              normed=True, histtype="stepfilled",
              color=colors[i], alpha=0.7)
    plt.title(u"Доход от " + _stock)
    plt.xlim(-0.15, 0.15)
    plt.xlabel(u'Значение')
    plt.ylabel(u'Плотность')

plt.tight_layout()
plt.suptitle(u"Гистограммы доходов в день от акций", size=14);
```

Далее выполним вывод на основе апостериорного среднего дохода и апостериорной матрицы ковариации. Получившееся апостериорное распределение приведено на рис. 6.14.

```
obs = pm.MvNormal("observed returns", mu, inv_cov_matrix, observed=True,
                  value=returns)
```

```
model = pm.Model([obs, mu, inv_cov_matrix])
mcmc = pm.MCMC()

mcmc.sample(150000, 100000, 3)
```

```
[Output]:

[****************100%******************] 150000 of 150000 complete
```

```
figsize(12.5,4)

# Сначала исследуем средний доход
mu_samples = mcmc.trace("returns")[:]

for i in range(4):
    plt.hist(mu_samples[:,i], alpha = 0.8 - 0.05*i, bins=30,
             histtype="stepfilled", normed=True,
             label="%s"%stock_returns.keys()[i])

plt.vlines(mu_samples.mean(axis=0), 0, 500, linestyle="--", linewidth=.5)

plt.title(u"Апостериорное распределение $\mu$, \n
           доход от акций в день")
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.legend();
```

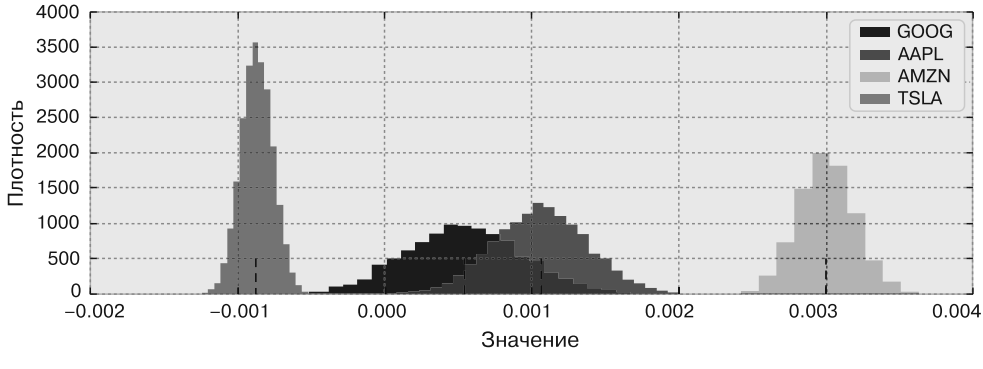

Рис. 6.14. Апостериорное распределение μ, доход от акций в день

Возможно, вы не сразу заметили, что эти переменные на целый порядок меньше, чем наши априорные распределения для них. Для примера приведем эти апостериорные распределения в том же масштабе, что и исходные априорные распределения (рис. 6.15).

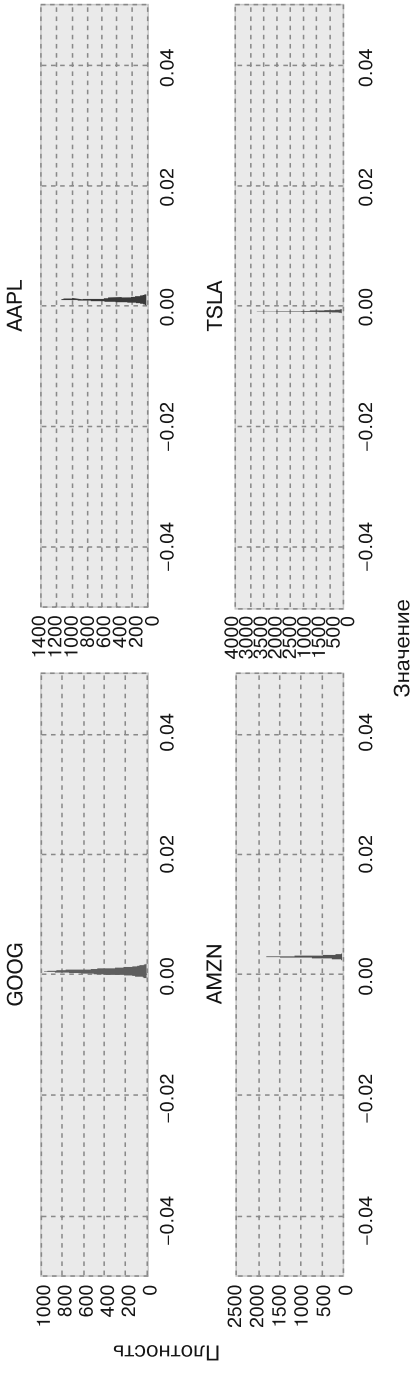

Рис. 6.15. Апостериорное распределение дохода от акций в день

```
figsize(11.0,3)
for i in range(4):
    plt.subplot(2,2,i+1)
    plt.hist(mu_samples[:,i], alpha=0.8 - 0.05*i, bins=30,
             histtype="stepfilled", normed=True, color=colors[i],
             label="%s"%stock_returns.keys()[i])
    plt.title("%s"%stock_returns.keys()[i])
    plt.xlim(-0.15, 0.15)

plt.title(u"Апостериорное распределение дохода от акций в день")
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.tight_layout()
```

Почему так происходит? Помните, я говорил, что у финансовых данных очень-очень низкое соотношение «сигнал — шум»? Это означает среду, в которой вывод требует существенно бóльших усилий. Нужно с осторожностью относиться к этим результатам, чтобы не переоценить их значение. Обратите внимание, что (см. рис. 6.14) в точке 0 все распределения больше нуля, а значит, дохода от акций может и не быть. Далее на результаты повлияли субъективные априорные распределения. С точки зрения фондового менеджера это хорошо, так как результаты отражают его/ее актуальные убеждения относительно акций, хотя с нейтральной точки зрения такой результат может считаться слишком субъективным.

На рис. 6.16 показаны апостериорная матрица корреляции и стандартное отклонение. Важно отметить, что распределение Уишарта моделирует обратную матрицу ковариации, так что для получения матрицы ковариации необходимо ее инвертировать. Мы также нормируем ее для получения матрицы корреляции. Поскольку на практике невозможно построить графики сотен матриц, мы удовлетворимся обобщением апостериорного распределения матриц корреляции в виде демонстрации *средней апостериорной матрицы корреляции* (mean posterior correlation matrix), то есть поэлементного математического ожидания апостериорного распределения матрицы. На практике его можно вычислить путем усреднения выборок из апостериорного распределения.

```
inv_cov_samples = mcmc.trace("inv_cov_matrix")[:]
mean_covariance_matrix = np.linalg.inv(inv_cov_samples.mean(axis=0))

def cov2corr(A):
    """
    Преобразование матрицы ковариации в матрицу корреляции
    """
    d = np.sqrt(A.diagonal())
    A = ((A.T/d).T)/d
    return A

plt.subplot(1,2,1)
plt.imshow(cov2corr(mean_covariance_matrix), interpolation="none",
           cmap = plt.cm.hot)
```

```
plt.xticks(np.arange(4), stock_returns.keys())
plt.yticks(np.arange(4), stock_returns.keys())
plt.colorbar(orientation="vertical")
plt.title(u"(Средняя апостериорная) матрица корреляции")

plt.subplot(1,2,2)
plt.bar(np.arange(4), np.sqrt(np.diag(mean_covariance_matrix)),
        color="#348ABD", alpha=0.7)
plt.xticks(np.arange(4) + 0.5, stock_returns.keys());
plt.title(u"(Средние апостериорные) дисперсии дохода от акций в день")
plt.xlabel(u'Вид акций')
plt.ylabel(u'Дисперсия')

plt.tight_layout();
```

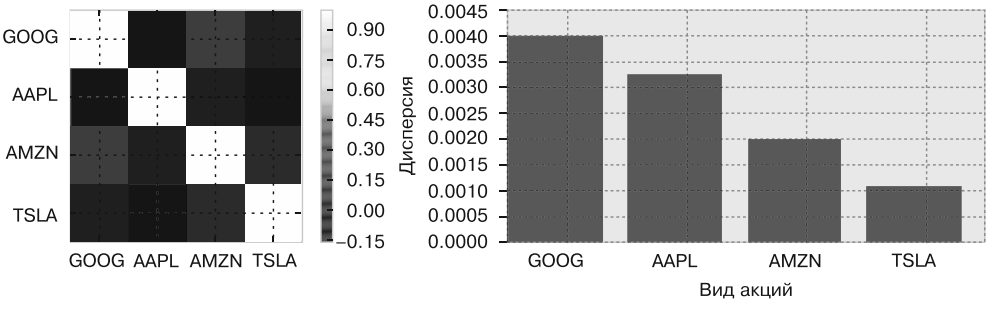

Рис. 6.16. *Слева:* (Средняя апостериорная) матрица корреляции. *Справа:* (Средние апостериорные) дисперсии дохода от акций в день

Исходя из рис. 6.16 довольно вероятно, что волатильность TSLA ниже среднего (это вполне очевидно из графика доходов). Матрица корреляции демонстрирует, что сильной корреляции не наблюдается, хотя, возможно, корреляция между GOOG и AMZN несколько выше (около 0,30).

При байесовском анализе фондового рынка можно воспользоваться оптимизацией на основе средних значений и дисперсий случайных величин (особо подчеркну: ни в коем случае *не* применяйте оптимизацию на основе средних значений и дисперсий для частотных точечных оценок!) и найти минимум. В результате этого можно найти оптимальное соотношение между высоким доходом и высокой дисперсией. Если обозначить оптимальный вес $w_{\text{опт}}$, то функция максимизации будет выглядеть следующим образом:

$$w_{\text{опт}} = \max_{w} \frac{1}{N}\left(\sum_{i=0}^{N} \mu_i^T w - \frac{\lambda}{2} w^T \Sigma_i w\right),$$

где μ_i и Σ_i — i-е апостериорные оценки среднего дохода и матрицы ковариации. Это еще один пример оптимизации функции потерь.

6.5.3. Советы от профи по поводу распределения Уишарта

В предыдущем примере распределение Уишарта сработало отлично. К сожалению, это не всегда так. Проблема в том, что оценка матрицы ковариации размером $N \times N$ требует оценки $\frac{1}{2}N(N-1)$ неизвестных величин. Это большое число даже при умеренно больших N. Я пробовал выполнять имитационное моделирование, аналогичное вышеприведенному, для $N = 23$ видов акций и в итоге сдался, когда понял, что моему алгоритму MCMC придется оценивать как минимум $23 \times 11 = 253$ дополнительные неизвестные величины (плюс другие интересные для этой задачи неизвестные величины). Весьма непростая задача для MCMC. По сути, для MCMC она означает обход более чем 250-мерного пространства. А ведь задача поначалу представлялась такой несложной. Вот несколько советов.

1. Используйте сопряженность, где это возможно.
2. Используйте хорошее начальное значение. Какое начальное значение окажется хорошим? Так выборочная матрица ковариации же! Обратите внимание, что речь не идет об эмпирической байесовской оценке; параметры априорного распределения не меняются, лишь модифицируется начальное значение MCMC. Из-за неустойчивости численного решения лучше округлить значения с плавающей точкой в выборочной матрице ковариации на несколько десятичных знаков (из-за неустойчивости могут возникать несимметричные матрицы, к которым PyMC очень плохо относится).
3. По возможности предоставить как можно больше знаний о предметной области в форме априорных распределений. Я подчеркиваю: «по возможности». Вероятнее всего, у вас не будет возможности оценить (априорно) каждую из $\frac{1}{2}N(N-1)$ неизвестных величин. В этом случае см. совет 4.
4. Используйте в качестве параметра априорного распределения эмпирические байесовские оценки, то есть выборочную матрицу ковариации.
5. В тех задачах, где N очень велико, никакие ухищрения не помогут. Лучше задайтесь вопросом: так ли уж вам важна *каждая* из корреляций? Вероятно, нет. Далее спросите себя: правда ли вам очень, *очень* важны корреляции? Возможно, что нет. В сфере финансового анализа имеет смысл установить неофициальную иерархию наиболее интересующих нас показателей: прежде всего, хорошая оценка μ; во-вторых, дисперсии вдоль диагонали матрицы ковариации; и в наименьшей степени — корреляции. Возможно, лучше проигнорировать $\frac{1}{2}N(N-1)(N-2)$ корреляций и сосредоточиться на более важных неизвестных величинах.

6.6. Сопряженные априорные распределения

Напомню, что априорное бета-распределение при биномиальных данных приводит к апостериорному бета-распределению. Наглядно это можно представить следующим образом:

$$\overbrace{\text{Beta}}^{\text{априорное распределение}} \cdot \overbrace{\text{Binominal}}^{\text{данные}} = \overbrace{\text{Beta}}^{\text{апостериорное распределение}} .$$

Обратите внимание на бета-распределение с обеих сторон этого уравнения — нет, сократить их нельзя, это не *настоящее* уравнение, а просто модель. Это свойство очень удобно — благодаря ему можно избежать использования MCMC, поскольку апостериорное распределение известно в аналитической форме. Это упрощает вывод и анализ. На этом упрощенном способе основывается алгоритм байесовских бандитов. К счастью, существует целое семейство распределений с подобными свойствами.

Пусть случайная переменная X взята (или по крайней мере мы так считаем) из известного распределения f_α, где α — возможно, неизвестные параметры f (которое может быть нормальным распределением или биномиальным распределением и т. п.). Для конкретного распределения f_α может существовать априорное распределение p_β, такое что:

$$\overbrace{p_\beta}^{\text{априорное распределение}} \cdot \overbrace{f_\alpha(X)}^{\text{данные}} = \overbrace{p_{\beta'}}^{\text{апостериорное распределение}} ,$$

где β' — другой набор параметров для того же априорного распределения p. Априорное распределение p, удовлетворяющее этой модели, называется *сопряженным априорным распределением* (conjugate prior). Как я уже упоминал, они удобны с вычислительной точки зрения, поскольку позволяют избежать приближенного вывода с помощью MCMC и перейти непосредственно к апостериорному распределению. Звучит заманчиво, правда?

К сожалению, не все так просто. Существует несколько проблем с сопряженными априорными распределениями.

- Сопряженные априорные распределения не являются объективными. Следовательно, их можно использовать только тогда, когда требуется субъективное априорное распределение. Причем нет гарантий, что сопряженное априорное распределение удастся подогнать под субъективное мнение специалиста-практика.
- Сопряженные априорные распределения обычно предусмотрены для простых, одномерных задач. Для задач с более сложными структурами надежды найти

сопряженное априорное распределение нет. Для упомянутых более простых моделей в «Википедии» есть удобная таблица сопряженных априорных распределений [10].

На самом деле сопряженные априорные распределения полезны только своим математическим удобством, так как упрощают переход от априорного распределения к апостериорному. Я рассматриваю их лишь как ловкий математический трюк, никакой новой информации об изучаемой проблеме нам не дающий.

6.7. Априорное распределение Джеффриса

Ранее я упоминал, что объективные априорные распределения редко бывают на самом деле объективными. В частности, имелось в виду, что априорное распределение не должно смещать апостериорные оценки. Разумным вариантом в этом смысле выглядит плоское распределение, ведь при нем вероятность всех значений одинакова.

Однако плоское распределение не является *инвариантным по отношению к преобразованиям*. Что это значит? Пусть у нас есть случайная переменная $\mathbf{X}$ из распределения Bernoulli(θ). Зададим априорное распределение для $p(\theta) = 1$. Оно показано на рис. 6.17.

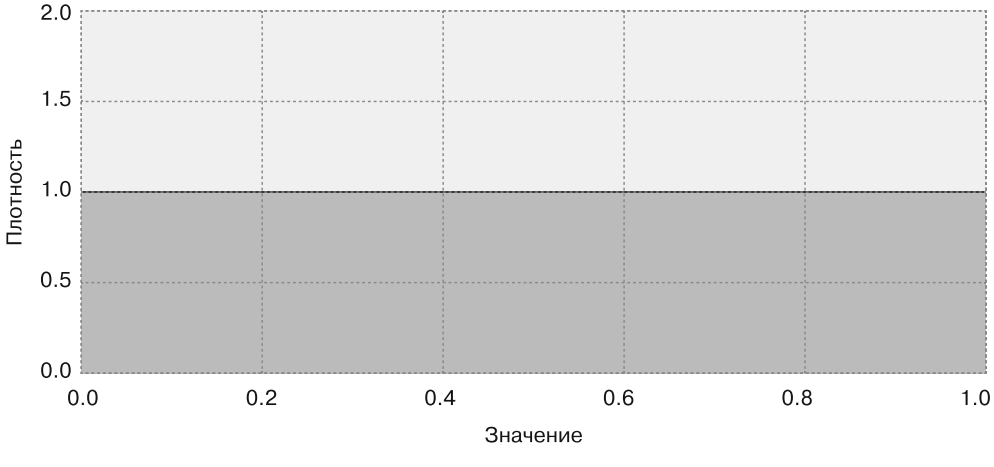

Рис. 6.17. Априорное распределение θ

```
figsize(12.5, 5)

x = np.linspace(0.000, 1, 150)
y = np.linspace(1.0, 1.0, 150)
```

```
lines = plt.plot(x, y, color="#A60628", lw=3)
plt.fill_between(x, 0, y, alpha=0.2, color=lines[0].get_color())
plt.autoscale(tight=True)
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.ylim(0, 2);
```

Произведем теперь преобразование θ с помощью функции $\Psi = \log\left(\frac{\theta}{1-\theta}\right)$. Эта функция просто растягивает θ вдоль вещественной оси. Какова вероятность различных значений Ψ при нашем преобразовании?

```
figsize(12.5, 5)

psi = np.linspace(-10, 10, 150)
y = np.exp(psi) / (1 + np.exp(psi))**2
lines = plt.plot(psi, y, color="#A60628", lw = 3)
plt.fill_between(psi, 0, y, alpha = 0.2, color = lines[0].get_color())
plt.autoscale(tight=True)
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.ylim(0, 1);
```

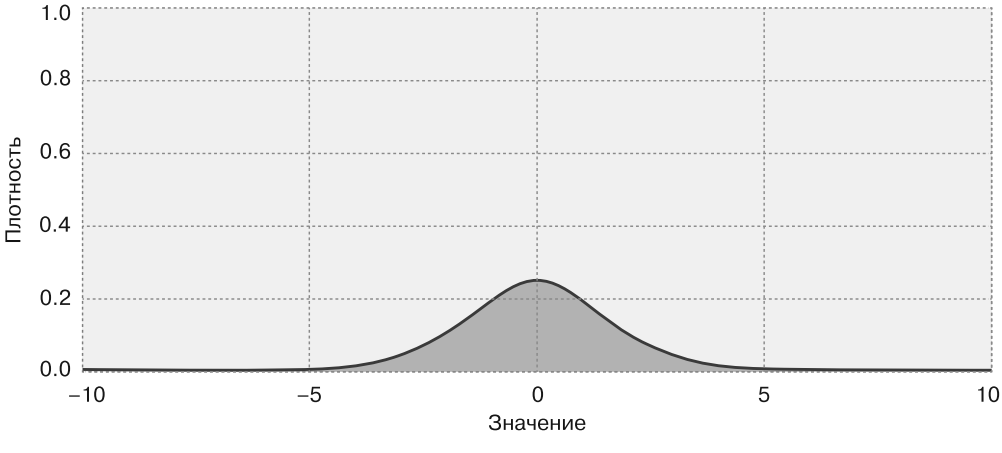

Рис. 6.18. Априорное распределение

Как можно видеть из рис. 6.18, функция перестает быть плоской! Оказалось, что плоские априорные распределения все же несут в себе какую-то информацию. Смысл распределений Джеффриса состоит в создании априорных распределений, которые не начнут вдруг нести какую-то информацию в случае преобразования соответствующих переменных. Этот вопрос выходит за рамки данной книги, но существует много другой литературы по априорным распределениям Джеффриса.

6.8. Влияние априорных распределений при изменении N

В главе 1 я говорил, что чем больше объем имеющихся наблюдений (данных), тем меньшее значение имеет априорное распределение. Это интуитивно понятно. В конце концов, в основе априорного распределения лежит исходная информация, так что постепенно новая информация отодвигает старую на задний план. Достаточное количество данных позволяет также подавить влияние априорного распределения; если априорное распределение по своей сути ошибочно, то благодаря самокорректирующейся природе данных мы получим *менее ошибочное*, а в конце концов и *правильное* апостериорное распределение.

Опишем это на математическом языке. Прежде всего вспомним теорему Байеса из главы 1, связывающую априорное распределение с апостериорным.

> Апостериорное распределение параметра θ при заданном наборе данных $\mathbf{X}$ можно записать в следующем виде:
>
> $$p(\theta \mid \mathbf{X}) \propto \underbrace{p(\mathbf{X} \mid \theta)}_{\substack{\text{мера} \\ \text{правдоподобия}}} \cdot \overbrace{p(\theta)}^{\substack{\text{априорная} \\ \text{вероятность}}}$$
>
> или в чаще применяемой логарифмической шкале:
>
> $$\log(p(\theta \mid \mathbf{X})) = c + L(\theta; \mathbf{X}) + \log(p(\theta))$$
>
> Логарифмическая мера правдоподобия $L(\theta; \mathbf{X}) = \log p(X|\theta)$, будучи функцией от данных, *растет пропорционально размеру выборки*, в то время как плотность априорного распределения — нет. Следовательно, при росте размера выборки абсолютное значение $L(\theta; \mathbf{X})$ растет, в то время как $\log(p(\theta))$ остается неизменным (при фиксированном значении θ); значит, при росте размера выборки $L(\theta; \mathbf{X})$ сильнее влияет на сумму $L(\theta; \mathbf{X}) + \log(p(\theta))$.

У этих выкладок есть интересное, хотя и незаметное на первый взгляд следствие. При росте размера выборки влияние выбранного априорного распределения снижается. Следовательно, вывод будет сходиться независимо от выбранного априорного распределения, конечно, при одних и тех же областях ненулевой вероятности.

На рис. 6.19 мы визуализируем вышесказанное. Мы посмотрим на сходимость двух апостериорных распределений с биномиальным параметром θ: одно — с плоским априорным распределением, а второе — с априорным распределением, смещенным к 0. По мере роста размера выборки апостериорные вероятности, а следовательно, и вывод сходятся[1].

---

[1] Для более аккуратного отображения субграфиков (на этом и некоторых ранее приведенных графиках), без пересечений, можно добавить в конце следующий вызов: plt.tight_layout().

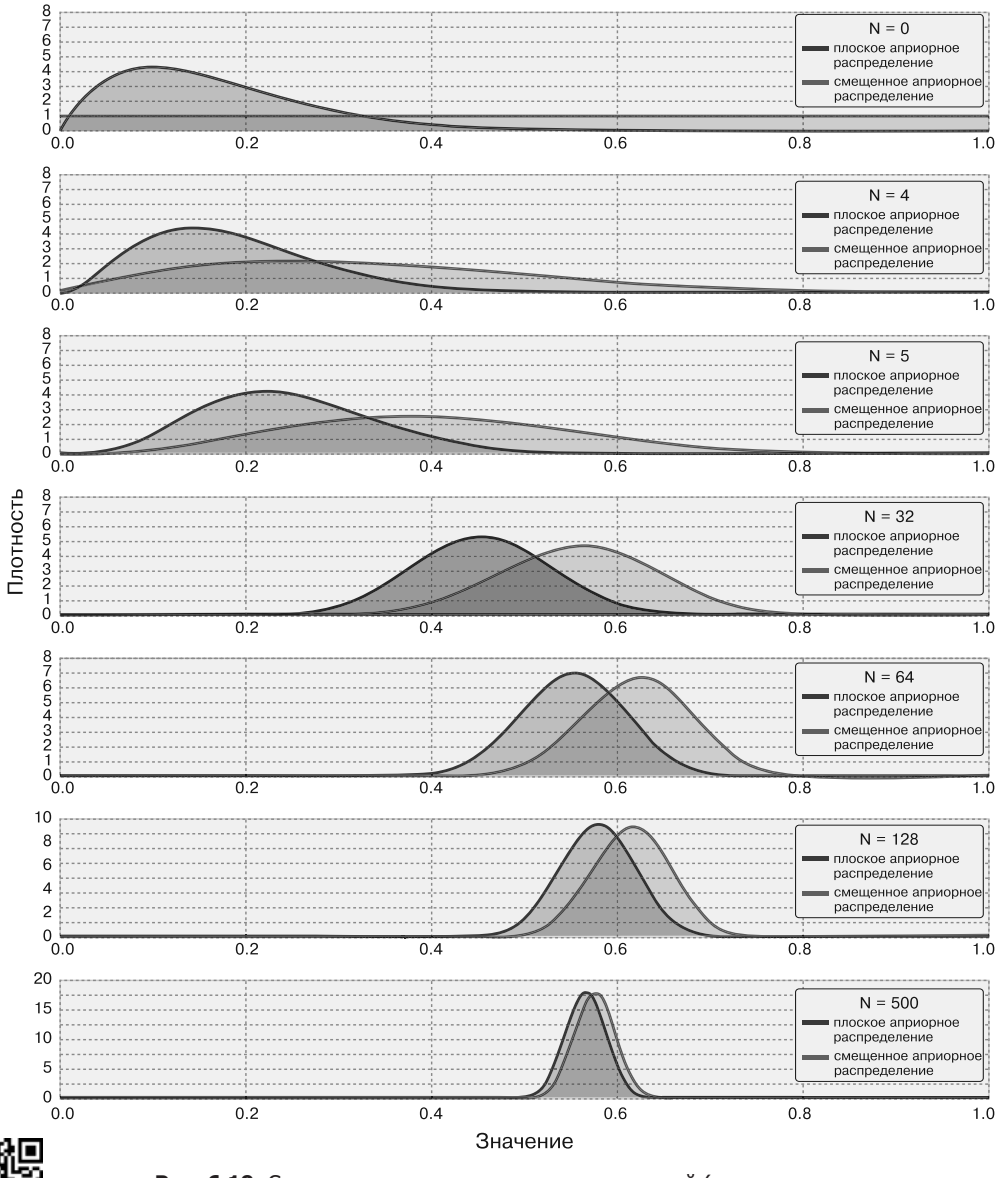

Рис. 6.19. Сходимость апостериорных распределений (с различными априорными распределениями) по мере наблюдения все новой информации

```
figsize(12.5, 15)
p = 0.6
beta1_params = np.array([1.,1.])
```

```
beta2_params = np.array([2,10])
beta = stats.beta

x = np.linspace(0.00, 1, 125)
data = pm.rbernoulli(p, size=500)

plt.figure()
for i,N in enumerate([0, 4, 8, 32, 64, 128, 500]):
    s = data[:N].sum()
    plt.subplot(8, 1, i+1)
    params1 = beta1_params + np.array([s, N-s])
    params2 = beta2_params + np.array([s, N-s])
    y1,y2 = beta.pdf(x, *params1), beta.pdf(x, *params2)
    plt.plot(x, y1, label=u"плоское априорное распределение", lw =3)
    plt.plot(x, y2, label=u"смещенное априорное распределение", lw= 3)
    plt.fill_between(x, 0, y1, color="#348ABD", alpha=0.15)
    plt.fill_between(x, 0, y2, color="#A60628", alpha=0.15)
    plt.legend(title="N=%d"%N)
    plt.vlines(p, 0.0, 7.5, linestyles="--", linewidth=1)
    plt.xlabel(u'Значение')
    plt.ylabel(u'Плотность')
    plt.title(u"Сходимость апостериорных распределений (с различными
              априорными распределениями) по мере наблюдения все
              новой информации")
```

Учтите, что не все апостериорные распределения «забудут» априорное так быстро. Этот пример призван лишь продемонстрировать, что *в конце концов* априорное распределение будет забыто. Именно вследствие этой «забываемости» априорного распределения по мере нашего погружения во все новые данные как байесовский, так и частотный вывод в конечном итоге сходятся.

6.9. Выводы

В этой главе мы повторно проанализировали использование априорных распределений. Априорное распределение оказалось еще одним дополнением к модели, причем требующим тщательного выбора. Зачастую априорное распределение рассматривают как самую слабую и одновременно самую сильную сторону байесовского вывода. Первое потому, что сама идея выбора априорного распределения означает субъективизм и личные мнения экспертов, а последнее потому, что обеспечивает исключительную гибкость моделей для любых данных.

На тему априорных распределений были написаны сотни научных статей, исследования в этой сфере существенно расширили границы байесовского анализа. Не следует недооценивать их важность, в том числе на практике. Я надеюсь, что вы почерпнули из этой главы некоторые эвристические алгоритмы выбора достойных априорных распределений.

6.10. Приложение

6.10.1. Байесовская точка зрения на линейную регрессию со штрафом

Между линейной регрессией методом наименьших квадратов со штрафом и байесовскими априорными распределениями существует очень интересная взаимосвязь. Линейная регрессия со штрафом представляет собой задачу оптимизации вида:

$$\arg\min_{\beta}(Y - X\beta)^T(Y - X\beta) + f(\beta)$$

для некоей функции f, которая обычно является нормой, например $\|\cdot\|_p^p$. При $p = 1$ получается модель, в которой штраф накладывается на абсолютное значение коэффициентов β. При $p = 2$ получается гребневая регрессия (ridge regression), в которой штраф накладывается на квадрат коэффициентов β.

Опишем сначала вероятностную интерпретацию линейной регрессии методом наименьших квадратов. Обозначим переменную отклика Y, а матрицу признаков — X. Стандартная линейная модель имеет вид:

$$Y = X\beta + \varepsilon,$$

где $\varepsilon \sim \text{Normal}(\mathbf{0}, \sigma\mathbf{I})$, $\mathbf{0}$ — вектор нулей, а $\mathbf{I}$ — единичная матрица. Проще говоря, наблюдаемая переменная Y представляет собой линейную функцию от X (с коэффициентами β) плюс некую шумовую составляющую. Неизвестная величина, которую нужно определить, — β. Воспользуемся следующим свойством нормальных случайных переменных:

$$\mu' + \text{Normal}(\mu, \sigma) \sim \text{Normal}(\mu' + \mu, \sigma)$$

и перепишем эту линейную модель в виде:

$$Y = X\beta + \text{Normal}(\mathbf{0}, \sigma\mathbf{I});$$
$$Y = \text{Normal}(X\beta, \sigma\mathbf{I}).$$

В вероятностной системе обозначений зададим распределение вероятности переменной Y как $f_Y(\gamma \mid \beta)$ и вспомним функцию плотности для нормальной случайной переменной (см. [11]):

$$f_Y(Y \mid \beta, X) = \text{Правдоподобие}(\beta \mid X, Y) = \frac{1}{\sqrt{2\pi\sigma}}\exp\left(\frac{1}{2\sigma^2}(Y - X\beta)^T(Y - X\beta)\right).$$

Это функция правдоподобия для β. Прологарифмируем:

$$l(\beta) = K - c(Y - X\beta)^T(Y - X\beta),$$

где константы K и c больше 0. Согласно методу максимального правдоподобия нам нужно найти максимум для β:

$$\hat{\beta} = \arg\max_{\beta} -(Y - X\beta)^T (Y - X\beta).$$

Или, что эквивалентно, найти минимум отрицательного по отношению к нему выражения:

$$\hat{\beta} = \arg\min_{\beta} -(Y - X\beta)^T (Y - X\beta).$$

Это хорошо знакомое нам уравнение линейной регрессии методом наименьших квадратов. Следовательно, мы показали, что решение задачи линейной регрессии наименьших квадратов — такое же, как и задачи поиска максимального правдоподобия, при условии, что шум распределен по нормальному закону. Далее мы обобщим этот результат и придем к линейной регрессии со штрафом, выбрав подходящее априорное распределение для β.

При известной мере правдоподобия мы можем включить в предыдущее уравнение априорное распределение для β и вывести уравнение для апостериорного распределения:

$$P(\beta|Y, X) = \text{Правдоподобие}(\beta|X, Y)p(\beta),$$

где $p(\beta)$ — априорная вероятность для компонентов β. Рассмотрим несколько интересных априорных распределений.

1. Если не включать в уравнение никакого явного члена для априорного распределения, это будет эквивалентно включению неинформативного априорного распределения, $P(\beta) \propto 1$. Его можно рассматривать как равномерно распределенное по всему множеству чисел.
2. Если у нас есть основания полагать, что компоненты β не слишком велики, то можно предположить, что:

   $$\beta \sim \text{Normal}(\mathbf{0}, \lambda\mathbf{I}).$$

   Полученная в результате апостериорная функция плотности вероятности для β *пропорциональна*:

   $$\exp\left(\frac{1}{2\sigma^2}(Y - X\beta)^T (Y - X\beta)\right)\exp\left(\frac{1}{2\lambda^2}\beta^T\beta\right).$$

   Берем логарифм от этого выражения, объединяем и переопределяем константы и получаем:

   $$l(\beta) \propto K - (Y - X\beta)^T(Y - X\beta) - \alpha\beta^T\beta,$$

Мы получили функцию, которую нужно максимизировать (напомню, что точка, в которой достигается максимум апостериорного распределения, — это MAP, апостериорный максимум):

$$\hat{\beta} = \arg\max_{\beta} \; -(Y - X\beta)^T (Y - X\beta) - \alpha\beta^T\beta.$$

Это эквивалентно минимизации соответствующего выражения со знаком минус. Перепишем также $\beta^T\beta = \|\beta\|_2^2$:

$$\hat{\beta} = \arg\min_{\beta} \; -(Y - X\beta)^T (Y - X\beta) + \alpha\|\beta\|_2^2.$$

Это в точности соответствует описанию гребневой регрессии. Следовательно, можно сделать вывод, что гребневая регрессия соответствует MAP линейной модели с нормально распределенными погрешностями и нормальным априорным распределением компонентов β.

3. Аналогично, если предположить априорно, что компоненты β распределены по закону Лапласа:

$$f_\beta(\beta) \propto \exp(-\lambda\|\beta\|_1)$$

и пройти по вышеизложенной схеме, мы получим выражение:

$$\hat{\beta} = \arg\min_{\beta} \; -(Y - X\beta)^T (Y - X\beta) + \alpha\|\beta\|_1,$$

представляющее собой LASSO-регрессию. Приведу несколько важных замечаний относительно эквивалентности этих методов. Возникающая в результате использования LASSO-регуляризации разреженность никак не связана с высокой вероятностью значений 0 в априорном распределении; на самом деле все как раз наоборот. Разреженность для β возникает в результате сочетания функции $\|\cdot\|_1$ и использования MAP. Впрочем, априорное распределение, конечно, вносит свой вклад в общее снижение значений коэффициентов до 0. Интересное обсуждение этого вопроса можно найти в [12].

Для повторения байесовской линейной регрессии вы можете взглянуть на пример с финансовыми потерями из главы 5.

6.10.2. Выбор вырожденного априорного распределения

Если в какой-либо области вероятность в априорном распределении ненулевая, то в апостериорном распределении вероятность может оказаться какой угодно. Что произойдет, если мы присвоим нулевую вероятность области, к которой действительно относится фактическое значение? Проведем небольшой эксперимент для демонстрации такой ситуации. Пусть мы наблюдаем данные, распределенные

по закону Бернулли, и хотели бы получить оценку значения p (вероятности успешного исхода).

```
p_actual = 0.35
x = np.random.binomial(1, p_actual, size=100)
print x[:10]
```

```
[Output]:

[0 0 0 0 1 0 0 0 1 1]
```

Выберем неудачное априорное распределение для p, скажем, Uniform(0,5, 1). Это априорное распределение задает для фактического значения 0,35 вероятность 0. Посмотрим, что произойдет при выводе.

```
import pymc as pm

p = pm.Uniform('p', 0.5, 1)
obs = pm.Bernoulli('obs', p, value=x, observed=True)

mcmc = pm.MCMC([p, obs])
mcmc.sample(10000, 2000)
```

```
[Output]:

[-----------------100%-----------------] 10000 of 10000 complete in 0.7 sec
```

```
p_trace = mcmc.trace('p')[:]
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.hist(p_trace, bins=30, histtype='stepfilled', normed=True);
```

На рис. 6.20 можно видеть, что апостериорное распределение вдребезги разбивается о нижнюю границу априорного распределения. Это значит, что фактическое значение, вероятно, меньше 0,5. Подобная картина апостериорного распределения — хороший индикатор проблем с априорными допущениями.

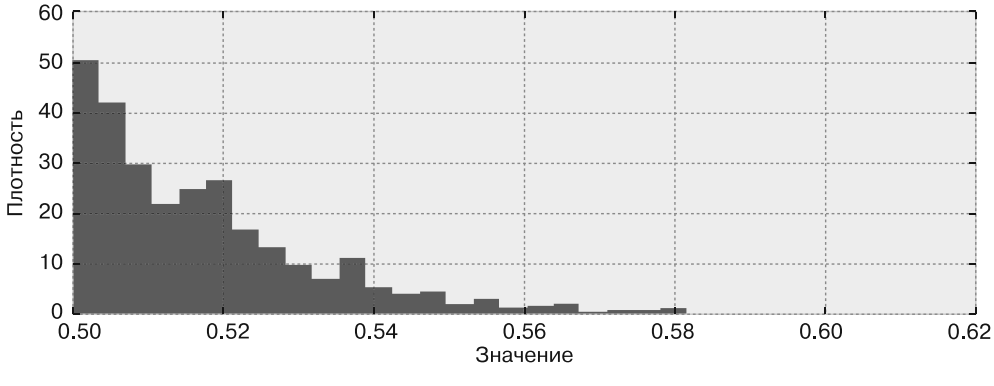

Рис. 6.20. Апостериорное распределение неизвестной величины p с априорным распределением Uniform(0,5, 1)

6.11. Библиография

1. *Gelman A., Shalizi C. R.* Philosophy and the Practice of Bayesian Statistics // British Journal of Mathematical and Statistical Psychology, 66 (2013): 8–38.
2. *Gelman A.* Prior Distributions for Variance Parameters in Hierarchical Models // Bayesian Analysis, 1, no. 3 (2006): 515–534.
3. *Scott S. L.* A Modern Bayesian Look at the Multi-Armed Bandit // Applied Stochastic Models in Business and Industry, 26 (2010): 639–658.
4. *Kuleshov V., Precup D.* Algorithms for the Multi-Armed Bandit Problem // Journal of Machine Learning Research, 1 (2000): 1–48.
5. *Neufeld J.* Reddit's 'Best' Comment Scoring Algorithm as a Multi-Armed Bandit Task // Simple ML Hacks, April 9, 2013. http://simplemlhacks.blogspot.com/2013/04/reddits-best-comment-scoring-algorithm.html.
6. *Oakley J. E., Daneshkhah A., O'Hagan A.* Nonparametric Elicitation Using the Roulette Method. http://www.tonyohagan.co.uk/academic/pdf/elic-roulette.pdf.
7. Eliciting Priors from Experts // Cross Validated. http://stats.stackexchange.com/questions/1/eliciting-priors-from-experts.
8. *Oakley J. E.* Eliciting Univariate Probability Distributions. http://www.jeremy-oakley.staff.shef.ac.uk/Oakley_elicitation.pdf.
9. *Taleb N. N.* The Black Swan: The Impact of the Highly Improbable. New York: Random House, 2007.
10. Сопряженное априорное распределение. Википедия, Свободная энциклопедия. https://ru.wikipedia.org/wiki/Сопряжённое_априорное_распределение.
11. Нормальное распределение. Википедия, Свободная энциклопедия. https://ru.wikipedia.org/wiki/Нормальное_распределение.
12. *Starck, J.-L., Donoho D. L., Fadili M. J., Rassat A.* Sparsity and the Bayesian Perspective // Astronomy and Astrophysics, February 18, 2013.

7 A/B-тестирование

7.1. Введение

Одна из целей специалистов-статистиков и исследователей данных — бороться за чистоту экспериментов, а один из лучших инструментов исследователя данных — хорошо организованный эксперимент по сплит-тестированию. Мы уже сталкивались со сплит-тестами. В главе 2 был показан байесовский анализ A/B-теста для частоты конверсий на сайте. В этой главе мы обобщим приведенный анализ на новые области.

7.2. Краткое резюме вышеприведенного A/B-тестирования конверсий

В методе A/B-тестирования рассматривается идеальная гипотетическая вселенная. Генеральная совокупность в ней идентична контрольной, но отличается применением какого-либо фактора, на счет которого можно отнести все наблюдаемые различия. На практике создавать новые вселенные сложновато, так что придется ограничиться разбиением достаточного количества выборок на две группы, которые бы *приближенно* имитировали подобные гипотетические вселенные.

Напомню пример из главы 2: существует два варианта дизайна сайта, A и B. При входе пользователя на сайт ему демонстрируется случайным образом один из вариантов дизайна, после чего записываются его действия. Когда наберется достаточное число посетителей, набор данных анализируется на предмет какой-либо метрики (в случае сайтов речь идет обычно о покупках или регистрации). Например, взгляните на следующие показатели:

```
visitors_to_A = 1300
visitors_to_B = 1275

conversions_from_A = 120
conversions_from_B = 125
```

Фактически нас интересует вероятность конверсии для каждого из сайтов. С точки зрения бизнеса желательно, чтобы эта вероятность была как можно выше. Так что наша цель — определить, у какого из сайтов, A или B, вероятность конверсии выше.

Для этого мы моделируем вероятность конверсии для сайта A и сайта B. При моделировании вероятности в качестве априорного распределения в нашем случае хорошо подойдет бета-распределение. (Почему? Потому что оно ограничено значениями от 0 до 1, что совпадает с диапазоном значений, допустимым для вероятностей.) Количество посетителей и количество конверсий распределены биномиально: для сайта A при 1300 *испытаний* число *успешных конверсий* равно 120. Как вы помните из главы 6, априорное бета-распределение и биномиальные наблюдения сопряжены; это значит, что никакого MCMC не требуется!

Если априорное распределение Beta(α_0, β_0) и наши наблюдения включают N испытаний и X успешных конверсий, то апостериорное распределение будет иметь вид Beta($\alpha_0 + X$, $\beta_0 + N - X$). С помощью встроенной функции `beta` из библиотеки SciPy можно непосредственно выполнить выборку из этого распределения.

Пусть априорное распределение имеет вид Beta(1, 1). Как вы помните, оно идентично равномерному распределению на отрезке [0, 1].

```
from scipy.stats import beta
alpha_prior = 1
beta_prior = 1

posterior_A = beta(alpha_prior + conversions_from_A,
                   beta_prior + visitors_to_A - conversions_from_A)

posterior_B = beta(alpha_prior + conversions_from_B,
                   beta_prior + visitors_to_B - conversions_from_B)
```

Далее нам нужно определить, в какой группе вероятность конверсии выше. Для этого аналогично MCMC мы возьмем выборки из апостериорного распределения и оценим вероятность того, что выборки из апостериорного распределения A больше выборок из распределения B. Для генерации выборок воспользуемся методом `rvs`.

```
samples = 20000 # Для повышения точности число выборок должно быть большим.
samples_posterior_A = posterior_A.rvs(samples)
samples_posterior_B = posterior_B.rvs(samples)

print (samples_posterior_A > samples_posterior_B).mean()
```

```
[Output]:

0.31355
```

Как видим, вероятность того, что конверсия сайта A лучше, чем сайта B, равна 31 % (и наоборот, вероятность того, что конверсия сайта B лучше, чем сайта A, равна 69 %). Не слишком значительная разница. Если выполнить эксперимент снова при одинаковых веб-страницах, то вероятность будет близка к 50 %.

Можно визуализировать апостериорное распределение без гистограмм, с помощью метода `pdf`. На рис. 7.1 показаны апостериорные распределения конверсий сайтов A и B.

```
%matplotlib inline
from IPython.core.pylabtools import figsize
from matplotlib import pyplot as plt
figsize(12.5, 4)
plt.rcParams['savefig.dpi'] = 300
plt.rcParams['figure.dpi'] = 300

x = np.linspace(0,1, 500)
plt.plot(x, posterior_A.pdf(x), label=u'апостериорное распределение A')
plt.plot(x, posterior_B.pdf(x), label=u'апостериорное распределение B')
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.title(u"Апостериорные распределения частоты конверсий
          веб-страниц $A$ и $B$")
plt.legend();
```

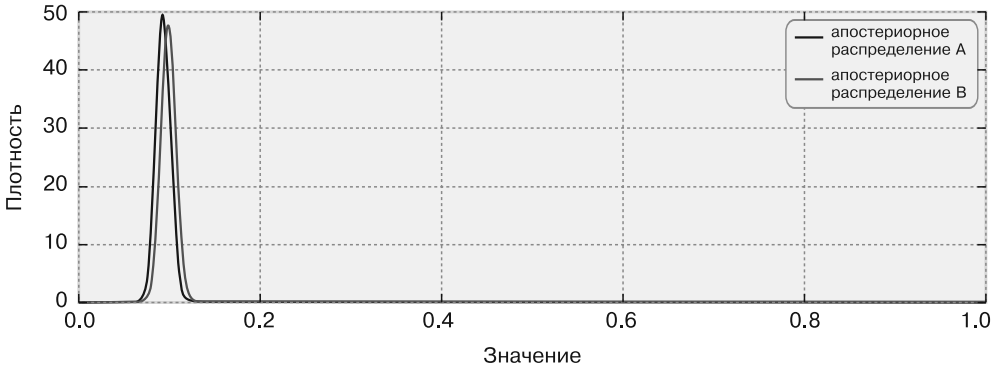

Рис. 7.1. Апостериорные распределения частоты конверсий веб-страниц A и B

На рис. 7.2 мы увеличим интересующую нас область.

Тестирование конверсии популярно благодаря своей простоте: наблюдаемое значение бинарное, так что анализ очень прост. Но что будет, если пользователь сможет выбирать один из множества путей, каждый из которых повлечет различные последствия? Мы изучим этот вопрос в следующем разделе.

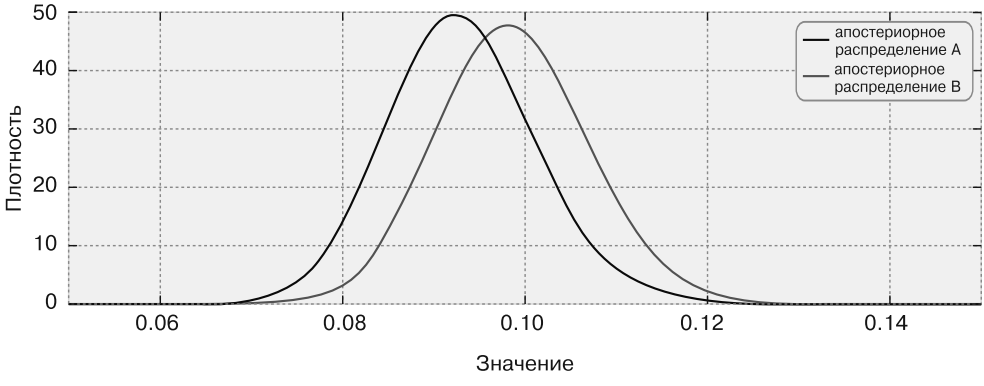

Рис. 7.2. Апостериорные распределения частоты конверсий веб-страниц A и B крупным планом

7.3. Добавляем линейную функцию потерь

Работающие в Интернете компании ставят своей целью не только повышение количества регистраций на сайтах, но и оптимизацию выбираемых пользователями планов подписки. Например, с точки зрения бизнеса выгодно, чтобы пользователь выбирал более дорогой план подписки из двух или более вариантов.

Пусть пользователям демонстрируются два различных варианта страницы с расценками и мы хотели бы оценить *ожидаемую выручку* (expected revenue) из расчета на один показ. В предыдущем A/B-тестировании нас интересовало только, зарегистрируется ли пользователь на сайте; теперь же мы хотели бы знать, какую выручку можем ожидать.

7.3.1. Анализ ожидаемой выручки

Забудем пока об A/B-тестировании и поговорим об анализе в случае одного стиля веб-страницы. В абсолютно понятном мире, где нам известно все, можно было бы вычислить математическое ожидание для нашей вымышленной компании:

$$E[R] = 79p_{79} + 49p_{49} + 25p_{25} + 0p_0,$$

где p_{79} — вероятность выбора плана подписки за \$79 и т. д. Я также включил сюда вымышленный план подписки за \$0 для учета тех пользователей, которые *не* выберут никакого плана. Сумма вероятностей равна 1:

$$p_{79} + p_{49} + p_{25} + p_0 = 1.$$

Следующий шаг — оценка этих вероятностей. Воспользоваться бета/биномиальной моделью для каждой из вероятностей нельзя, поскольку они коррелируют между собой; их сумма должна равняться 1. Например, если вероятность p_{79} высока, то остальные вероятности должны быть низкими. Нам придется моделировать все эти вероятности вместе.

Существует обобщение биномиального распределения, которое называется *мультиномиальным распределением* (multinomial distribution). Оно реализовано как в PyMC, так и в NumPy, но я воспользуюсь вариантом из NumPy. В следующем коде задается значение вектора вероятностей `P`, который определяет вероятности попадания в соответствующую корзину. Если задать длину `P` равной 2 (и сделать так, чтобы сумма вероятностей равнялась 1), то мы получим привычное биномиальное распределение:

```
from numpy.random import multinomial
P = [0.5, 0.2, 0.3]
N = 1
print multinomial(N, P)
```

```
[Output]:

[1 0 0]
```

```
N = 10
print multinomial(N, P)
```

```
[Output]:

[4 3 3]
```

Наши наблюдения для страницы подписок соответствуют мультиномиальному распределению, в котором значения вектора вероятностей `P` нам неизвестны.

Существует также и обобщение бета-распределения. Оно носит название *распределения Дирихле* (Dirichlet distribution) и возвращает вектор положительных чисел, дающих в сумме 1. Длина этого вектора определяется длиной входного вектора, значения которого аналогичны параметрам априорного распределения.

```
from numpy.random import dirichlet
sample = dirichlet([1,1]) # [1,1] эквивалентно распределению Beta(1,1)
print sample
print sample.sum()
```

```
[Output]:

[ 0.3591 0.6409]
1.0
```

```
sample = dirichlet([1,1,1,1])
```

```
print sample
print sample.sum()
```

```
[Output]:

[ 1.5935e-01 6.1971e-01 2.2033e-01 6.0750e-04]
1.0
```

К счастью, между распределением Дирихле и мультиномиальным распределением существует взаимосвязь, аналогичная взаимосвязи бета-распределения и биномиального распределения. Распределение Дирихле — распределение, сопряженное мультиномиальному! Это значит, что существуют точные формулы для апостериорных распределений неизвестных вероятностей. Если априорное распределение имеет вид Dirichlet (1, 1... 1), а наблюдения обозначены N_1, N_2... N_m, то апостериорное распределение имеет вид:

$$\text{Распределение}(1 + N_1, 1 + N_2 \ldots 1 + N_m).$$

Сумма выборок из этого апостериорного распределения всегда равна 1, так что их можно использовать в формуле математического ожидания из подраздела 7.3.1. Посмотрим на примере каких-либо данных. Пусть страницу просмотрели 1000 человек при следующих количествах подписок на различные планы:

```
N    = 1000
N_79 = 10
N_49 = 46
N_25 = 80
N_0  = N - (N_79 + N_49 + N_49)

observations = np.array([N_79, N_49, N_25, N_0])

prior_parameters = np.array([1,1,1,1])
posterior_samples = dirichlet(prior_parameters + observations, size=10000)

print "Две случайные выборки из апостериорного распределения:"
print posterior_samples[0]
print posterior_samples[1]
```

```
[Output]:

Две случайные выборки из апостериорного распределения:
[ 0.0165 0.0497 0.0638 0.8701]
[ 0.0123 0.0404 0.0694 0.878 ]
```

Можно построить график функции плотности вероятности этого апостериорного распределения:

```
for i, label in enumerate(['p_79', 'p_49', 'p_25', 'p_0']):
    ax = plt.hist(posterior_samples[:,i], bins=50,
                  label=label, histtype='stepfilled')

plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.title(u"Апостериорные распределения вероятности выбора различной\
          стоимости плана подписки")
plt.legend();
```

Как вы можете видеть на рис. 7.3, неопределенность относительно значений вероятностей все равно присутствует, так что она будет присутствовать и в математическом ожидании. Это нормально; мы получили апостериорное распределение математического ожидания. Для этого мы пропускаем все выборки из распределения Дирихле через приведенную ниже функцию `expected_revenue`.

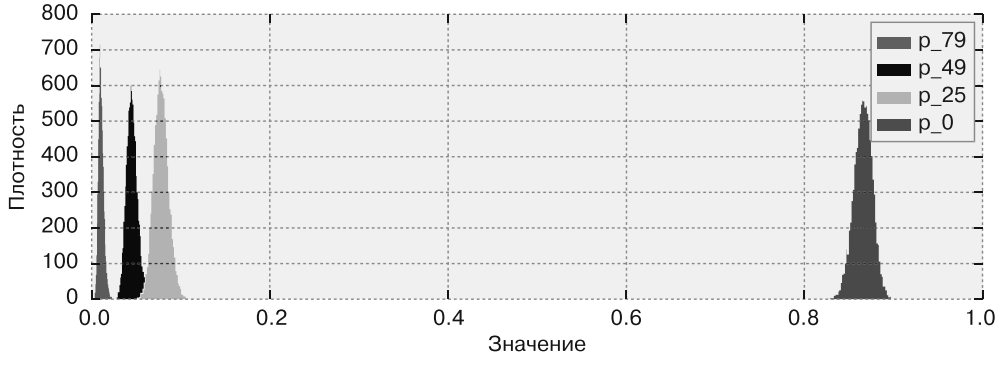

Рис. 7.3. Апостериорные распределения вероятности выбора различной стоимости плана подписки

Данный подход сильно напоминает использование функции потерь, и, по существу, именно это и происходит: мы оцениваем параметры, затем передаем их в функцию потерь, чтобы снова соотнести с реальным миром (рис. 7.4).

```
def expected_revenue(P):
    return 79*P[:,0] + 49*P[:,1] + 25*P[:,2] + 0*P[:,3]

posterior_expected_revenue = expected_value(posterior_samples)
plt.hist(posterior_expected_revenue, histtype='stepfilled',
         label=u'ожидаемая выручка', bins=50)
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.title(u"Апостериорные распределения ожидаемой выручки")
plt.legend();
```

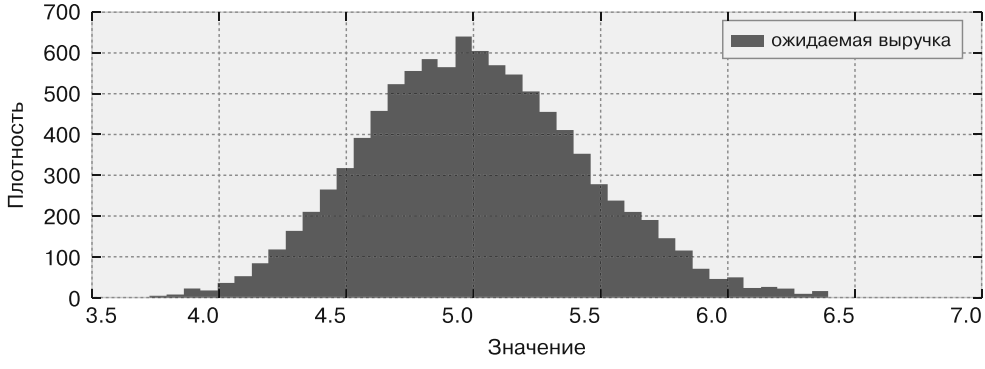

Рис. 7.4. Апостериорные распределения ожидаемой выручки

Как видно на рис. 7.4, ожидаемая выручка, вероятно, находится в диапазоне от $4 до 6 и вряд ли вне его.

7.3.2. Обобщение на случай A/B-эксперимента

Попробуем провести вышеприведенный анализ для случая двух веб-страниц, обозначенных A и B, для которых я создал следующие модельные данные (рис. 7.5):

```
N_A    = 1000
N_A_79 = 10
N_A_49 = 46
N_A_25 = 80
N_A_0  = N_A - (N_A_79 + N_A_49 + N_A_49)
observations_A = np.array([N_A_79, N_A_49, N_A_25, N_A_0])

N_B    = 2000
N_B_79 = 45
N_B_49 = 84
N_B_25 = 200
N_B_0  = N_B - (N_B_79 + N_B_49 + N_B_49)
observations_B = np.array([N_B_79, N_B_49, N_B_25, N_B_0])

prior_parameters = np.array([1,1,1,1])

posterior_samples_A = dirichlet(prior_parameters + observations_A,
                                size=10000)
posterior_samples_B = dirichlet(prior_parameters + observations_B,
                                size=10000)

posterior_expected_revenue_A = expected_revenue(posterior_samples_A)
```

```
posterior_expected_revenue_B = expected_revenue(posterior_samples_B)

plt.hist(posterior_expected_revenue_A, histtype='stepfilled',
         label=u'ожидаемая выручка сайта A', bins=50)
plt.hist(posterior_expected_revenue_B, histtype='stepfilled',
         label=u'ожидаемая выручка сайта B', bins=50, alpha=0.8)
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.title(u"Апостериорные распределения ожидаемой выручки для страниц $A$ и $B$")
plt.legend();
```

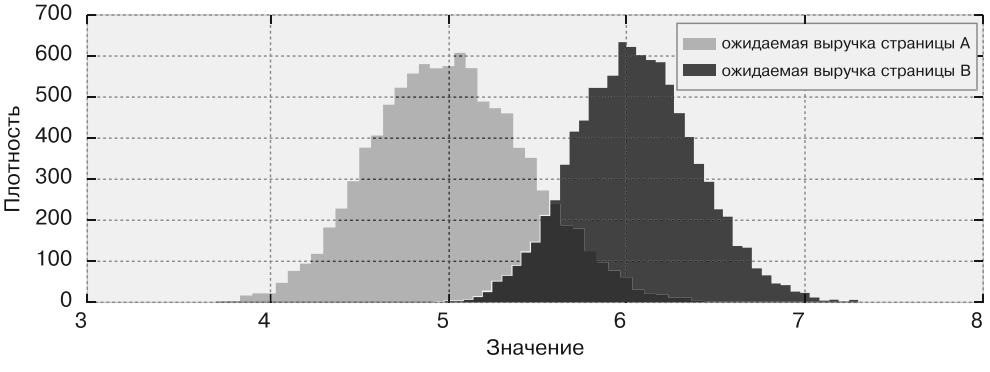

Рис. 7.5. Апостериорные распределения ожидаемой выручки для страниц A и B

Обратите внимание, насколько далеко на рис. 7.5 отстоят два апостериорных распределения, что означает существенное различие показателей двух наших веб-страниц. Средняя ожидаемая выручка страницы A почти на доллар меньше, чем у страницы B (на первый взгляд, не так уж и много, но это ведь из расчета *на один просмотр страницы*, что может сложиться в немалую сумму). Чтобы подтвердить существование подобного различия, можно взглянуть на вероятность того, что выручка для страницы A больше, чем для страницы B, как мы делали в предыдущем анализе конверсии.

```
p = (posterior_expected_revenue_B > posterior_expected_revenue_A).mean()
print "Вероятность того, что выручка страницы B больше, чем страницы A: %.3f"%p
```

```
[Output]:

Вероятность того, что выручка страницы B больше, чем страницы A: 0.965
```

Значение 96 % достаточно велико, чтобы быть статистически значимым, так что имеет смысл выбрать для промышленной эксплуатации страницу B.

Еще один интересный график — апостериорное различие выручки страниц, показанное на рис. 7.6. Его создание не требовало от нас дополнительных усилий, поскольку речь идет о байесовском анализе. Достаточно построить гистограмму различий двух апостериорных распределений ожидаемой выручки.

```
posterior_diff = posterior_expected_revenue_B -
                 posterior_expected_revenue_A

plt.hist(posterior_diff, histtype='stepfilled', color='#7A68A6',
         label=u'разница в выручке страниц B и A', bins=50)
plt.vlines(0, 0, 700, linestyles='--')
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.title(u"Апостериорное распределение дельты ожидаемой выручки \
          между страницами $A$ и $B$")
plt.legend();
```

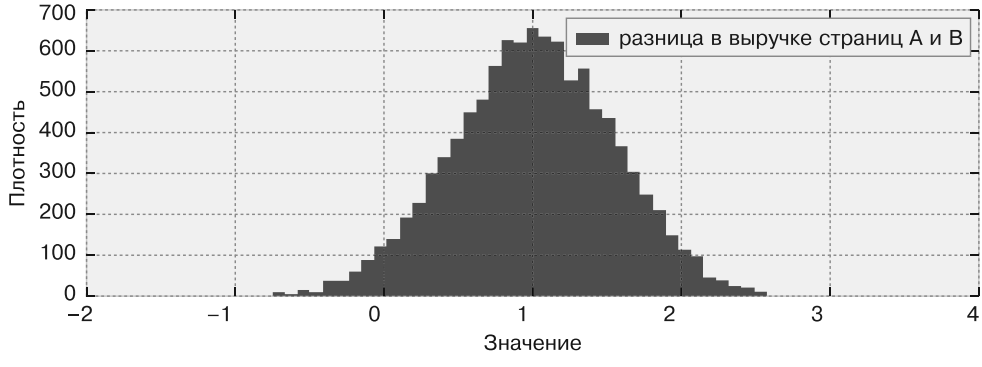

Рис. 7.6. Апостериорное распределение дельты ожидаемой выручки между страницами A и B

Исходя из этого апостериорного распределения, можно сказать, что вероятность разницы *более* чем $1 (а возможно, и больше) составляет примерно 50 %. Даже если наш выбор сайта B окажется ошибочным (что возможно), вероятно, много мы не потеряем: распределение практически не простирается далее –$0,50.

7.4. Выходим за рамки конверсий: тест Стьюдента

Вероятно, самый популярный в университетах статистический тест — *тест Стьюдента* (t-test). Обычный тест Стьюдента представляет собой частотный тест, предназначенный для определения того, далеко ли отклоняется выборочное среднее от

заранее заданного значения. Здесь мы воспользуемся байесовской версией теста Стьюдента, активно продвигаемой Джоном К. Крушке (John K. Kruschke). Эта модель носит название BEST (от англ. Bayesian estimation supersedes the t-test — «байесовское оценивание вытесняет тест Стьюдента»). Первоначальная статья Крушке [1] написана весьма доступным языком, и я очень рекомендую вам с ней ознакомиться.

7.4.1. Схема теста Стьюдента

Следуя нашему примеру для A/B-тестирования, предположим, что у нас есть набор данных относительно времени нахождения пользователя на тестовой странице. Причем данные не бинарные, а непрерывные. Например, создадим с помощью следующего кода модельные данные:

```
N = 250
mu_A, std_A = 30, 4
mu_B, std_B = 26, 7

# Создаем данные о длительности (в секундах) нахождения пользователя
# на каждой из страниц
durations_A = np.random.normal(mu_A, std_A, size=N)
durations_B = np.random.normal(mu_B, std_B, size=N)
```

Не забывайте, что на практике параметры из предыдущего блока кода нам неизвестны, мы видим только выводимые результаты:

```
print durations_A[:8]
print durations_B[:8]
```

```
[Output]:

[34.2695 28.4035 22.5516 34.1591 31.1951 27.9881 30.0798 30.6869]
[36.1196 19.1633 32.6542 19.7711 27.5813 34.4942 34.1319 25.6773]
```

Наша задача — определить, на какой из страниц, A или B, пользователи задерживаются дольше. Модель включает пять неизвестных величин, два параметра средних значений (обозначаемые μ), два параметра стандартных отклонений (обозначаемые σ) и один дополнительный параметр, специфичный для теста Стьюдента: ν (произносится «ню»). Параметр ν задает, насколько вероятны в наших данных большие аномальные значения. Согласно модели BEST априорные распределения для неизвестных величин следующие.

1. Параметры μ_A и μ_B имеют в качестве источника нормальное распределение с априорным средним значением, равным среднему значению объединенных данных для A и B, а априорное стандартное отклонение в 1000 раз превышает

стандартное отклонение объединенных данных A и B (это очень широкое и неинформативное априорное распределение).

```
import pymc as pm

pooled_mean = np.r_[durations_A, durations_B].mean()
pooled_std = np.r_[durations_A, durations_B].std()
tau = 1./np.sqrt(1000.*pooled_std) # В PyMC используется параметр
                                   # точности 1/sigma**2

mu_A = pm.Normal("mu_A", pooled_mean, tau)
mu_B = pm.Normal("mu_B", pooled_mean, tau)
```

2. Параметры σ_A и σ_B имеют в качестве источника равномерное распределение, ограниченное отрезком от 1/1000 до 1000 значений стандартного отклонения объединенных данных A и B (опять же, очень широкое неинформативное априорное распределение).

```
std_A = pm.Uniform("std_A", pooled_std/1000., 1000.*pooled_std)
std_B = pm.Uniform("std_B", pooled_std/1000., 1000.*pooled_std)
```

3. Наконец, ν оценивается на основе смещенного экспоненциального распределения с параметром, равным 29. Более подробно мотивы подобного выбора можно найти в приложении A к [1]. Интересный нюанс модели BEST — параметр ν один для обеих групп. Все станет понятнее из нижеприведенной схемы.

```
nu_minus_1 = pm.Exponential("nu-1", 1./29)
```

В целом наша модель выглядит так, как показано на рис. 7.7 (взятом из [2]). Завершаем сбор элементов модели воедино (рис. 7.8):

```
obs_A = pm.NoncentralT("obs_A", mu_A, 1.0/std_A**2, nu_minus_1 + 1,
                       observed=True, value=durations_A)
obs_B = pm.NoncentralT("obs_B", mu_B, 1.0/std_B**2, nu_minus_1 + 1,
                       observed=True, value=durations_B)

mcmc = pm.MCMC([obs_A, obs_B, mu_A, mu_B, std_A, std_B, nu_minus_1])
mcmc.sample(25000,10000)
```

```
[Output]:

[-----------------100%-----------------] 25000 of 25000 complete in 16.6 sec
```

```
mu_A_trace, mu_B_trace = mcmc.trace('mu_A')[:], mcmc.trace('mu_B')[:]
std_A_trace, std_B_trace = mcmc.trace('std_A')[:], mcmc.trace('std_B')[:]
nu_trace = mcmc.trace("nu-1")[:] + 1

figsize(12,8)
```

```
def _hist(data, label, **kwargs):
    return plt.hist(data, bins=40, histtype='stepfilled',
                    alpha=.95, label=label, **kwargs)

ax = plt.subplot(3,1,1)
_hist(mu_A_trace,'A')
_hist(mu_B_trace,'B')
plt.legend()
plt.title(u'Апостериорные распределения $\mu$')

ax = plt.subplot(3,1,2)
_hist(std_A_trace, 'A')
_hist(std_B_trace, 'B')
plt.legend()
plt.title(u'Апостериорные распределения $\sigma$')

ax = plt.subplot(3,1,3)
_hist(nu_trace,'', color='#7A68A6')
plt.title(ur'Апостериорное распределение $\nu$')
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.tight_layout();
```

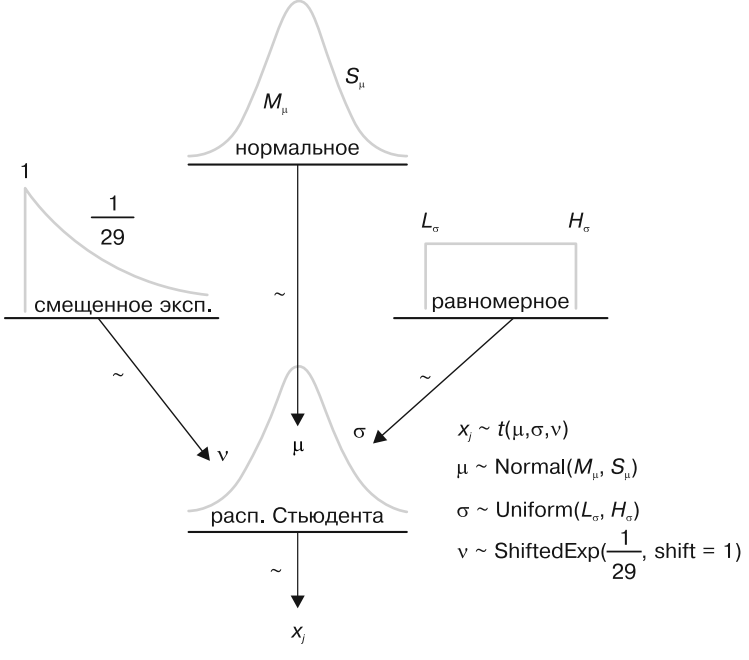

Рис. 7.7. Графическое представление модели BEST [2]

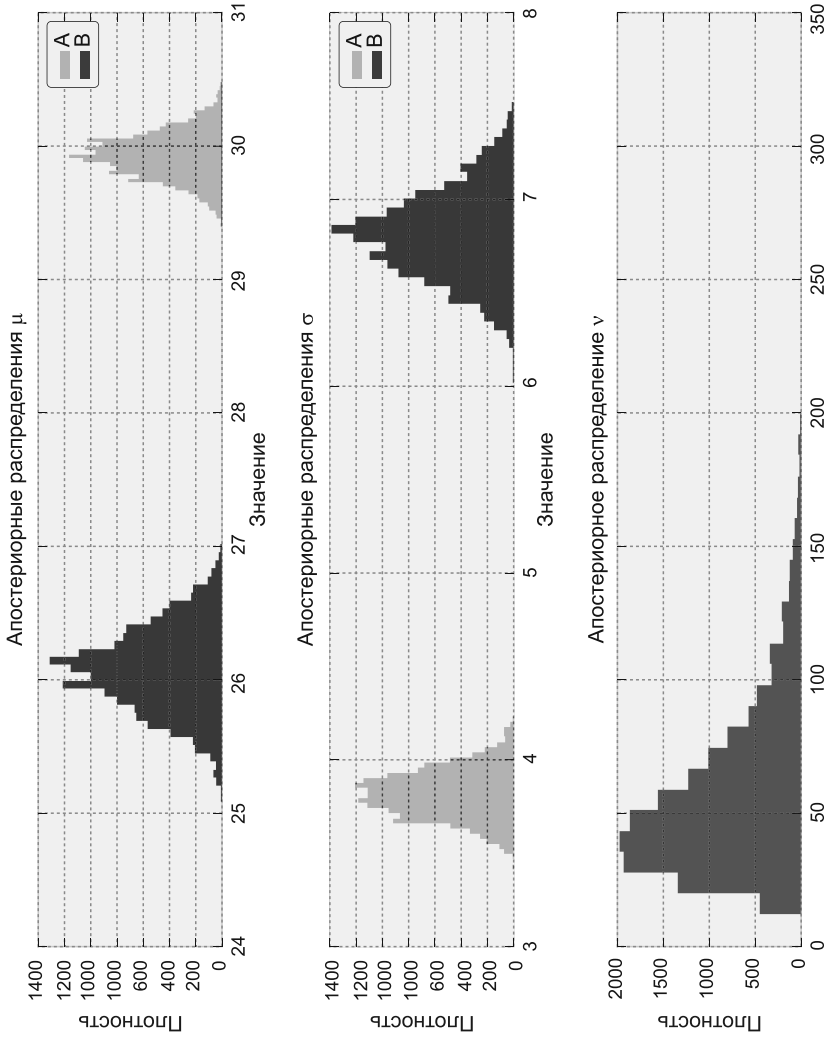

Рис. 7.8. Апостериорные распределения неизвестных параметров модели BEST

Из рис. 7.8 видно, что между двумя группами есть четкое различие (так и было задумано, конечно). В верхней части на рис. 7.8 приведены графики неизвестных μ_1 и μ_2. Второй график — график σ_1 и σ_2. Как видим, на странице A не только пользователи в среднем проводят больше времени, но и волатильность просмотров этой страницы ниже (поскольку для страницы A ниже стандартное отклонение). Далее на основе этих апостериорных распределений можно вычислить различия между группами, величины эффекта и т. д.

В модели BEST хорошо то, что ее можно оформить в виде аккуратной функции лишь с несколькими небольшими модификациями.

7.5. Оценка показателя роста

Лиц, принимающих решения на основе A/B-тестирования, часто интересует величина такого показателя, как *рост* (increase). Это неправильно, и я считаю, что подобное означает перепутать *непрерывную задачу* с *бинарной*. Непрерывная задача состоит в измерении того, *насколько лучше* (диапазон возможных значений непрерывен), а бинарная задача — в определении того, *что лучше* (возможных значений только два). Проблема в том, что решение непрерывной задачи требует на порядки больше данных, чем решение бинарной, но компании хотят использовать решение бинарной задачи для решения задачи непрерывной. На самом деле большинство распространенных статистических тестов предназначено для решения лишь бинарных задач, что мы и делали в предыдущих разделах.

В любом случае бизнес-компании хотят получить ответ на оба вопроса. Давайте сначала выясним, чего делать *не* стоит. Пусть вы оцениваете частоту конверсии обеих групп с помощью обсуждавшихся выше методов. Заказчика интересует показатель относительного роста результата, иногда называемый *продвижением* (lift) эксперимента. Одна из возможностей — «наивно» вычислить средние значения обоих апостериорных распределений и определить относительный рост:

$$\frac{\hat{p}_A - \hat{p}_B}{\hat{p}_B}.$$

Это может привести к серьезным ошибкам. Прежде всего мы отбросили всю неопределенность относительно фактических значений p_A и p_B. Оценивая продвижение с помощью предыдущего уравнения, мы неявно предположили, что знаем точные значения этих величин. Практически всегда это приводит к серьезному завышению оценки значений, особенно когда p_A и p_B близки к 0. Именно поэтому иногда можно встретить смехотворные газетные заголовки вроде «Как один-единственный A/B-тест повысил конверсию на 336 %» [3] (кстати, это реальный заголовок!).

Проблема в том, что мы хотели бы сохранить неопределенность; статистика — это в конце концов моделирование неопределенности! Для этого мы передадим апостериорные распределения в функцию и получим в результате новое апостериорное распределение. Попробуем это в нашем заключительном A/B-тесте. Апостериорные распределения приведены на рис. 7.9.

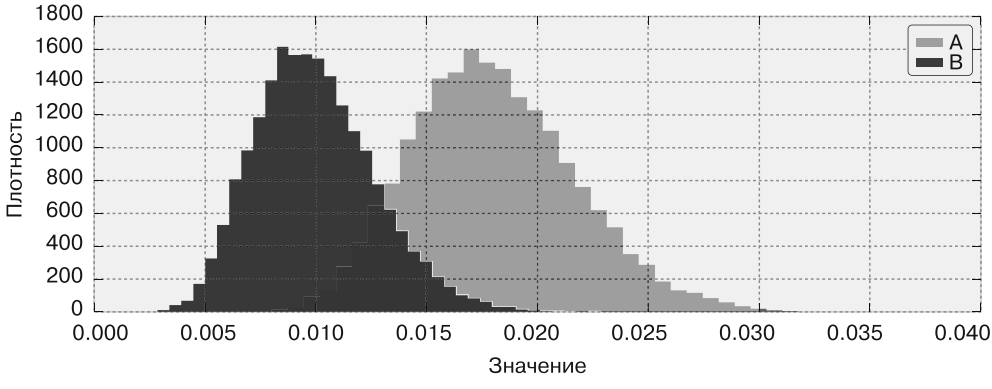

Рис. 7.9. Апостериорные распределения частоты конверсии веб-страниц A и B

```
figsize(12,4)

visitors_to_A = 1275
visitors_to_B = 1300

conversions_from_A = 22
conversions_from_B = 12

alpha_prior = 1
beta_prior = 1

posterior_A = beta(alpha_prior + conversions_from_A,
                   beta_prior + visitors_to_A - conversions_from_A)

posterior_B = beta(alpha_prior + conversions_from_B,
                   beta_prior + visitors_to_B - conversions_from_B)

samples = 20000
samples_posterior_A = posterior_A.rvs(samples)
samples_posterior_B = posterior_B.rvs(samples)

_hist(samples_posterior_A, 'A')
_hist(samples_posterior_B, 'B')
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
```

```
plt.title(u"Апостериорные распределения частоты конверсии\
          веб-страниц $A$ и $B$")
plt.legend();
```

Мы передали апостериорные распределения в функцию, вычисляющую попарно относительный рост. Полученное апостериорное распределение показано на рис. 7.10.

```
def relative_increase(a,b):
    return (a-b)/b

posterior_rel_increase = relative_increase(samples_posterior_A,
      samples_posterior_B)
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.title(u"Апостериорное распределение относительного продвижения частоты\
          конверсии веб-страницы $A$ по сравнению с частотой конверсии\
          страницы $B$")

_hist(posterior_rel_increase, 'relative increase', color='#7A68A6');
```

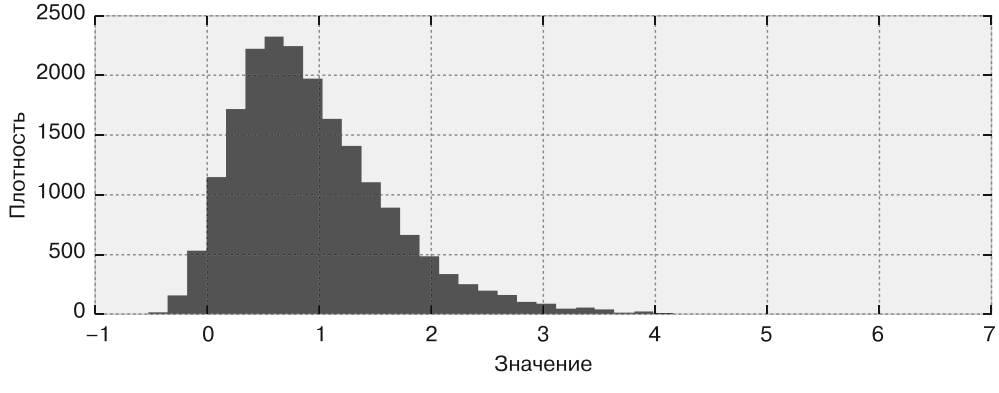

Рис. 7.10. Апостериорное распределение относительного продвижения частоты конверсии страницы A по сравнению с частотой конверсии страницы B

Из рис. 7.10 и этих новых вычислений можно видеть, что вероятность того, что относительный рост — 20 % или более, составляет 89 %. И даже больше: вероятность того, что относительный рост — 50 % или более, составляет 72 %.

```
print (posterior_rel_increase > 0.2).mean()
print (posterior_rel_increase > 0.5).mean()
```

```
[Output]:

0.89275
0.72155
```

А если бы мы наивно воспользовались точечными оценками:

$$\hat{p}_A = \frac{22}{1275} = 0,017,$$
$$\hat{p}_B = \frac{12}{1300} = 0,009,$$

то наша оценка относительного роста оказалась бы равна 87 % — вероятно, сильно завышенное значение.

7.5.1. Создание точечных оценок

Как я уже говорил, не слишком вежливо вручать распределение кому-то, особенно бизнес-заказчику, который ожидает получить одно значение. Что же делать? Мне кажется, есть три варианта.

1. Вернуть среднее значение апостериорного распределения относительного роста. Мне этот вариант не слишком нравится, и я попробую объяснить, почему. На рис. 7.10, *справа*, можно видеть длинный «хвост» возможных значений. Это значит, что распределение *асимметричное*. При асимметричном распределении на сводные статистические показатели слишком сильно влияет «хвост», следовательно, он окажется слишком сильно представлен в показателе и оценка *фактического* показателя относительного роста будет завышена.
2. Вернуть медианное значение апостериорного распределения относительного роста. В свете предыдущего обсуждения медиана — более подходящий вариант. Она отличается большей робастностью к асимметричным распределениям. Однако мне кажется, что на практике медиана дает все еще слишком завышенное значение.
3. Вернуть процентиль (< 50 %) апостериорного распределения относительного роста. Например, вернуть 30-й процентиль распределения. У такого варианта есть два желательных для нас свойства. Первое: он математически эквивалентен применению функции потерь к апостериорному распределению относительного роста, штрафующей завышенные оценки сильнее, чем заниженные, вследствие чего оценка будет достаточно осторожной. Во-вторых, по мере получения в результате эксперимента все новых и новых данных апостериорное распределение относительного роста все более и более сужается, а значит, все процентили постепенно сходятся к одной точке.

На рис. 7.11 я построил график трех сводных показателей:

```
mean = posterior_rel_increase.mean()
median = np.percentile(posterior_rel_increase, 50)
conservative_percentile = np.percentile(posterior_rel_increase, 30)

_hist(posterior_rel_increase,'', color='#7A68A6');
```

```
plt.vlines(mean, 0, 2500, linestyles='-.', label=u'среднее значение')
plt.vlines(median, 0, 2500, linestyles=':', label=u'медиана', lw=3)
plt.vlines(conservative_percentile, 0, 2500, linestyles='—',
           label=u'30-й процентиль')
plt.xlabel(u'Значение')
plt.ylabel(u'Плотность')
plt.title(u"Различные сводные показатели апостериорного распределения\
         относительного роста")
plt.legend();
```

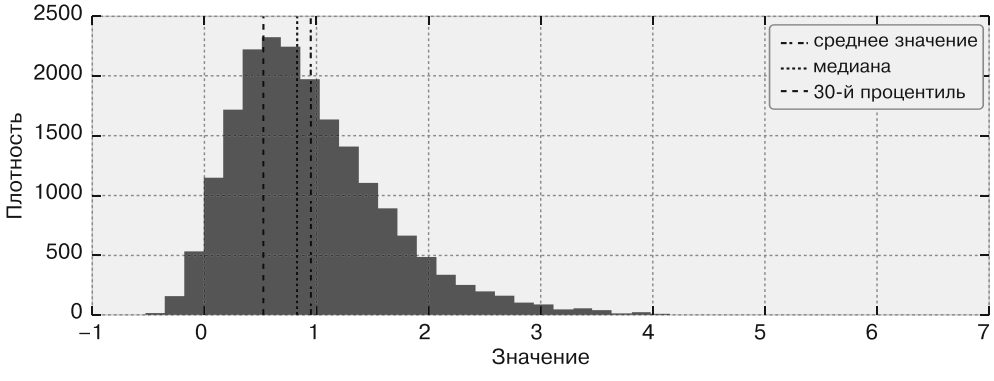

Рис. 7.11. Различные сводные показатели апостериорного распределения относительного роста

7.6. Выводы

В этой главе мы обсудили, как выглядит байесовское A/B-тестирование. Приведу два основных преимущества байесовского A/B-тестирования перед обычными методами.

1. Легко интерпретируемые вероятности. При байесовском анализе можно сразу ответить на вопрос: *какова вероятность того, что я ошибаюсь*? При использовании частотных методов ответить на него сложно или вообще невозможно.
2. Удобное применение функций потерь. В главе 5 мы видели, как функции потерь позволяют связывать абстрактные модели распределений вероятности с практическими задачами. В этой главе мы воспользовались линейной функцией потерь для определения ожидаемой выручки от просмотра страницы и еще одной функцией потерь для определения подходящей точечной оценки.

В этом, как и в других приложениях, байесовский анализ гибче и легче интерпретируем, чем другие методы. Кроме того, он требует лишь умеренных вычислительных затрат даже для сложных моделей. Я прогнозирую, что байесовское A/B-тестирование очень скоро станет гораздо более распространенным, чем традиционные методы.

7.7. Библиография

1. *Kruschke J. K.* Bayesian Estimation Supersedes the *t* test // Journal of Experimental Psychology: General, 142, № 2 (2013): 573–603.
2. Graphical representation of the BEST model by Rasmus Bååath, licensed under CC BY 4.0. http://www.sumsar.net/blog/2014/02/bayesian-first-aid-one-sample-t-test/.
3. *Sparks D.* How a Single A/B Test Increased Conversions by 336% [Case Study]. http://unbounce.com/a-b-testing/how-a-single-a-b-test-increased-conversions/.

Глоссарий

95%-ный байесовский доверительный интервал. Интервал из области значений апостериорной функции распределения вероятности, содержащий 95 % этого распределения.

95%-ное наименее правдоподобное значение. Нижняя граница 95%-ного байесовского доверительного интервала.

Автокорреляция. Мера взаимосвязи ряда чисел с ним же самим. Может принимать значения от 1 (абсолютная положительная автокорреляция) до –1 (абсолютная отрицательная автокорреляция).

Апостериорная вероятность. Скорректированное мнение относительно события A при известных данных X. Обозначается $P(A|X)$. См. *Априорная вероятность*.

Априорная вероятность. Обозначаемое $P(A)$ мнение относительно события A, предшествующее включению в модель информации относительно него. См. *Апостериорная вероятность*.

Байесианство. Статистическая парадигма, при которой вероятность рассматривается как мера уверенности в том, что событие произойдет.

Байесовская точечная оценка. Результат работы функции, вычисляющей сводный показатель апостериорного распределения.

Байесовские p-значения. Сводные показатели модели, аналогичные p-значениям из частотного подхода.

Бета-распределение. Распределение, задающее случайные переменные со значениями в диапазоне от 0 до 1 и потому часто применяемое для моделирования вероятностей и количественных соотношений.

Бинарная задача. Задача, при которой оценивается, какое (из двух возможных значений) лучше. Ср. *Непрерывная задача*.

Детерминистическая переменная. Переменная, принимающая неслучайные значения при известных значениях ее переменных-предков. Ср. *Стохастическая переменная*.

Дискретная случайная переменная. Переменная, которая может принимать значения лишь из заданного списка. Примером может служить рейтинг фильма. Ср. *Непрерывная случайная переменная*; *Случайная переменная смешанного типа*.

Доход в день. Относительные изменения вложенного капитала за один операционный биржевой день.

Критерий согласия. Мера того, насколько хорошо статистическая модель удовлетворяет наблюдаемым данным.

Непрерывная задача. Задача, при которой оценивается, насколько один результат лучше другого (в рамках непрерывного диапазона возможных значений). Ср. *Бинарная задача*.

Непрерывная случайная переменная. Переменная, которая может принимать произвольные точные значения, например отражающая величину температуры или скорости. Ср. *Дискретные случайные переменные*; *Случайная переменная смешанного типа*.

Объективное априорное распределение. Такое априорное распределение, при котором наибольшее влияние на апостериорное распределение оказывают данные. См. также *Плоское априорное распределение*; *Субъективное априорное распределение*.

Ожидаемое суммарное сожаление. Возможное среднее суммарное сожаление на основе большого числа испытаний для задачи о многоруких бандитах. Вычисляется путем усреднения результатов, полученных после множества прогонов алгоритма многоруких бандитов.

Ожидаемый доход в день. Ожидаемые относительные изменения вложенного капитала за один операционный биржевой день.

Переменная-потомок. Переменная, на которую влияет другая переменная. Ср. *Переменная-предок*.

Переменная-предок. Переменная, которая влияет на значение другой переменной (переменной-потомка).

Плоское априорное распределение. Равномерное распределение по всему диапазону значений неизвестной величины. Подразумевает равный вес для всех возможных значений.

Принцип безразличия. Идея, состоящая в том, что при выборе из n элементов, неразличимых между собой, каждый из элементов выбирается с вероятностью $1 / n$.

Разделительный график. Подход к визуализации данных, позволяющий сравнивать различные модели друг с другом.

Разреженное предсказание. Приближенная к 0 байесовская точечная оценка.

Распределение Бернулли. Бинарная случайная переменная, которая может принимать только значения 0 или 1.

Распределение Уишарта. Распределение, определенное на множестве всех положительно полуопределенных матриц.

Следы. Выборки, возвращаемые из апостериорного распределения алгоритмом MCMC (методом Монте-Карло по схеме марковской цепи).

Случайная переменная смешанного типа. Переменная, задающая вероятности как для дискретных, так и для непрерывных случайных переменных; сочетает в себе

оба этих типа переменных. Ср. *Непрерывная случайная переменная*; *Дискретная случайная переменная*.

Средняя апостериорная матрица корреляции. Поэлементное математическое ожидание апостериорного распределения матрицы, которое можно вычислить эмпирически путем усреднения значений выборок из апостериорного распределения.

Стохастическая переменная. Переменная, принимающая случайные значения даже при известных значениях переменных-предков. Ср. *Детерминистическая переменная*.

Субъективное априорное распределение. Априорное распределение, которое позволяет специалисту-статистику выражать свои взгляды. Ср. *Объективное априорное распределение*.

Тест Стьюдента. Частотный тест, предназначенный для определения отклонения выборочного среднего от заранее заданного значения.

Функция потерь. Функция от фактического значения параметра, служащая для измерения того, насколько (не)удачна текущая оценка.

Функция потерь на основе абсолютного значения ошибки. Разновидность функции потерь, растущая линейно пропорционально разнице между фактическим значением и оценкой. Часто применяется в машинном обучении и робастной статистике. См. также *Функция потерь*.

Функция потерь на основе среднеквадратичной ошибки. Разновидность функции потерь, которая растет пропорционально квадрату разницы. Используется при линейной регрессии, вычислении несмещенных сводных показателей и во многих сферах машинного обучения. См. также *Функция потерь*.

Частотный подход. Традиционная парадигма статистики, при которой вероятности определяются на основе частоты событий при большом количестве испытаний.

Эмпирический байесовский подход. Подход, сочетающий частотный и байесовский вывод благодаря выбору гиперпараметров с помощью частотных методов с последующим решением исходной задачи байесовскими методами.

Кэмерон Дэвидсон-Пайлон

Вероятностное программирование на Python: байесовский вывод и алгоритмы

Перевели с английского И. Пальти, К. Русецкий

Заведующая редакцией	*Ю. Сергиенко*
Руководитель проекта	*С. Давид*
Ведущий редактор	*Н. Гринчик*
Художественный редактор	*С. Маликова*
Корректор	*Е. Павлович*
Верстка	*Г. Блинов*

Изготовлено в России. Изготовитель: ООО «Прогресс книга».
Место нахождения и фактический адрес: 194044, Россия, г. Санкт-Петербург,
Б. Сампсониевский пр., д. 29А, пом. 52. Тел.: +78127037373.

Дата изготовления: 05.2019. Наименование: книжная продукция. Срок годности: не ограничен.

Налоговая льгота — общероссийский классификатор продукции ОК 034-2014, 58.11.12 —
Книги печатные профессиональные, технические и научные.

Импортер в Беларусь: ООО «ПИТЕР М», 220020, РБ, г. Минск, ул. Тимирязева, д. 121/3, к. 214, тел./факс: 208 80 01.

Подписано в печать 16.05.19. Формат 70×100/16. Бумага офсетная. Усл. п. л. 20,640. Тираж 700. Заказ 0000.

Отпечатано в ОАО «Первая Образцовая типография». Филиал «Чеховский Печатный Двор».
142300, Московская область, г. Чехов, ул. Полиграфистов, 1.
Сайт: www.chpk.ru. E-mail: marketing@chpk.ru
Факс: 8(496) 726-54-10, телефон: (495) 988-63-87

КНИГА-ПОЧТОЙ

ЗАКАЗАТЬ КНИГИ ИЗДАТЕЛЬСКОГО ДОМА «ПИТЕР» МОЖНО ЛЮБЫМ УДОБНЫМ ДЛЯ ВАС СПОСОБОМ:

- на нашем сайте: **www.piter.com**
- по электронной почте: **books@piter.com**
- по телефону: **(812) 703-73-74**

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ОПЛАТЫ:

- Наложенным платежом с оплатой при получении в ближайшем почтовом отделении.
- С помощью банковской карты. Во время заказа вы будете перенаправлены на защищенный сервер нашего оператора, где сможете ввести свои данные для оплаты.
- Электронными деньгами. Мы принимаем к оплате Яндекс.Деньги, Webmoney и Qiwi-кошелек.
- В любом банке, распечатав квитанцию, которая формируется автоматически после совершения вами заказа.

ВЫ МОЖЕТЕ ВЫБРАТЬ ЛЮБОЙ УДОБНЫЙ ДЛЯ ВАС СПОСОБ ДОСТАВКИ:

- Посылки отправляются через «Почту России». Отработанная система позволяет нам организовывать доставку ваших покупок максимально быстро. Дату отправления вашей покупки и дату доставки вам сообщат по e-mail.
- Вы можете оформить курьерскую доставку своего заказа (более подробную информацию можно получить на нашем сайте www.piter.com).
- Можно оформить доставку заказа через почтоматы,(адреса почтоматов можно узнать на нашем сайте www.piter.com).

ПРИ ОФОРМЛЕНИИ ЗАКАЗА УКАЖИТЕ:

- фамилию, имя, отчество, телефон, e-mail;
- почтовый индекс, регион, район, населенный пункт, улицу, дом, корпус, квартиру;
- название книги, автора, количество заказываемых экземпляров.

БЕСПЛАТНАЯ ДОСТАВКА:

- курьером по Москве и Санкт-Петербургу при заказе на сумму **от 2000 руб.**
- почтой России при предварительной оплате заказа на сумму **от 2000 руб.**