Обработка и анализ данных физического эксперимента

Конспект лекций

М.В. Корнилов

9 января 2025 г.

Предисловие

Настоящее учебное пособие, представляемое на суд читателя, имеет подзаголовок «конспект лекций», и этому есть две основных причины. Во-первых, основой послужил одноименный курс «Обработка и анализ данных физического эксперимента» для второго курса бакалавриата Факультета физики Высшей школы экономики. Этот курс создавался и существует с 2018 года с непосредственным участием автора, и предполагается, что текстовый конспект лекций послужит слушателям подспорьем при освоении теоретической части курса.

Во-вторых, конспект лекций, в отличии, например, от монографии, не претендует на полноту и охват информации по заданной теме, а является отправной точкой в изучении области. Основное внимание уделяется терминам, постановке задач и проблематике, которые встречаются в настоящее время в области обработки данных. Приводятся ссылки на литературу, заменой которой это пособие не служит. Большинство литературы — русскоязычные книги, в тех немногих случаях, когда книга ещё не переведена на русский язык используется ссылка на оригинальное издание. Также приведено несколько ссылок на оригинальные статьи из научных журналов, свободный доступ к которым обычно можно получить из локальной вычислительной сети университета.

При появлении в тексте нового термина он выделяется *курсивом*, и в сноске внизу страницы часто даётся его перевод на английский язык. Это оправдано преимущественно тем, что большинство популярных программных пакетов, где реализованы те или иные методы обработки данных, имеет документацию на английском языке. Предполагается, что читатель захочет применить изученные методы на практике, а для этого ему потребуется самостоятельно отыскать удобные для него готовые инструменты.

Благодарности

Существование настоящего учебного пособия стало возможно благодаря множеству разных людей и обстоятельств, но прежде всего благодарности заслуживают коллеги, которые в разное время делили бремя чтения курса «Обработка и анализ данных физического эксперимента» на Факультете физики Высшей школы экономики: Константин Маланчев, Евгений Мамонов, Владимир Бочарников и Фёдор Ратников.

Тяжкий труд по редактированию исходных рукописей взяла на себя Ирина Новожилова.

Компьютерная вёрстка осуществлена с применением пакета LaTeX.

Введение

Обработка и анализ данных — неотъемлемые части практически любого физического эксперимента. Под обработкой будем подразумевать преобразование значений одних физических величин, которые удаётся непосредственно измерить в рамках эксперимента[1], в значения других физических величин, которые представляют для нас интерес. А под анализом данных подразумевается «интеллектуальное переваривание» полученных значений: например, выявление каких-то новых зависимостей или проверка уже известных закономерностей на соответствие физической реальности.

Конечно, методы обработки и анализа данных требуются и применяются не только в физике, но и других науках, а взрывной рост цифровизации последних десятилетий привёл к выделению так называемой науки о данных в отдельную дисциплину. Обычно выделение этой области оправдывается тем, что применяемые в различных предметных областях подходы и методы в целом одни и те же.[2]

Предложенный курс обработки и анализа данных физического эксперимента задуман как своего рода мастерская, в которой можно освоить базовые инструменты для решения наиболее часто возникающих на практике задач. Конечно же, реальный мир обработки данных гораздо богаче: в разных областях физики встречаются свои специфичные задачи, для которых применяются соответствующие менее популярные методы. Охватить все это в рамках одного курса — проект сложный и малоперспективный, поскольку новые подходы и методы постоянно продолжают появляться, а старые — совершенствоваться. Важнее разобраться с тем, как применяются основные методы, и тогда остальные методы можно будет освоить по аналогии.

В современной практике все вычисления производятся на электронных вычислительных машинах (ЭВМ)независимо от того, был ли метод предложен до изобретения ЭВМ или после. По сравнению с ручными вычислениями это позволяет быстрее обработать бóльшие объёмы данных и снизить

[1] Для некоторых физических величин сопоставление со шкалой выглядит более наглядным, чем для других величин. Например, ускорение свободного падения может быть рассмотрено только в рамках экспериментов, где непосредственно измеряются интервалы времени, геометрические размеры и т.п.

[2] Скромно заметим, что очень долгое время именно физика была основным драйвером развития математических методов работы с данными.

трудозатраты. Тем не менее, эти вычисления не производятся мгновенно и есть строго обоснованные фундаментальные ограничения на скорость решения конкретных задач. В связи с этим необходимо оценивать количество ресурсов, которые потребуются для вычисления, и соотносить их с предполагаемыми методами, чтобы задача смогла быть решена на практике. Учитывая эту особенность, в данном курсе главы с рассмотрением конкретных методов предваряет глава, посвящённая алгоритмам и структурам данных. Такая организация материала позволяет сначала овладеть инструментарием по оценке требуемых вычислительных ресурсов и затем при изучении конкретных методов сразу представлять, насколько они требовательны к вычислительным ресурсам.

Каждая глава с описанием конкретных методов призвана дать новый полезный инструмент, совокупность которых образует «джентльменский набор», где имеется все необходимое для решения большинства задач по анализу данных. В рамках курса рассматриваются только самые популярные и актуальные задачи и методы их решения. Например, линейный метод наименьших квадратов (глава 2), анализ главных компонент (глава 3). Много методов обработки данных основаны на численных методах оптимизации, которым посвящена глава 4, где одновременно рассматриваются сопутствующие задачи, например, нелинейных наименьших квадратов. Метод максимального правдоподобия (глава 5) позволяет сформулировать некоторые подходы к анализу временных рядов (глава 6). Задачи проверки статистических гипотез рассматриваются в главе 7. Основные идеи методов машинного обучения рассматриваются в главе 8.

Напоследок стоит отметить, что в данном курсе мы имеем дело преимущественно с обратными задачами, суть которых состоит в том, чтобы исходя из существующих измерений определить параметры моделей физических систем. Способы же решения прямых задач, которые нужны, чтобы предсказать наблюдаемое поведение физических систем до опыта или вместо опыта, остаются за рамками данного опуса и рассматриваются в курсах, посвящённых вычислительным методам физики.

Глава 1

Анализ алгоритмов и структуры данных

Итак, условимся, что вычисления в рамках всех методов, рассматриваемых в данном курсе, производятся с применением электронных вычислительных машин (ЭВМ). Хотя некоторые методы были предложены задолго до изобретения ЭВМ, их использование предоставляет ряд неоспоримых преимуществ, прежде всего возможность безошибочно обрабатывать большие массивы информации.

Поскольку нас интересует не только принципиальная возможность решения тех или иных типовых задач, возникающих при обработке данных физических экспериментов, но и практические способы доведения расчётов до конкретного числа[1] или чисел (например, интересующих нас физических параметров) за разумное время, то мы неизбежно должны принять во внимание факт ограниченности вычислительной мощности (в том или ином смысле) доступных нам ЭВМ. Изначально может показаться, что такая низменная техническая проблема не достойна внимания исследователя-физика, однако, к сожалению, это не так. Пожалуй, даже начинающий физик-экспериментатор знает о том, что всё оборудование в его лаборатории имеет ту или иную стоимость[2], и соображения финансового плана часто играют не последнюю роль в планировании экспериментов. Аналогичная ситуация складывается и в области обработки данных: вычислительная мощность доступных нам ЭВМ ограничена либо в строгом смысле, т.е. где-то существует самый быстрый компьютер, когда-либо построенный человечеством, либо, чаще всего, в слабом смысле, когда оказывается, что стоимость требуемых ЭВМ составляет существенную часть доступного бюджета исследования. Иными словами, мы *вынуждены* проводить расчё-

[1] Под «конкретным числом» мы понимаем именно такое представление, которое сгодилось бы в качестве ответа на вопрос, например, «сколько топлива заливать в ракету?»

[2] И зачастую не маленькую стоимость, так как производство самых точных и хороших приборов требует самых совершенных из доступных человечеству технологий.

ты эффективно [3]. Данная глава целиком посвящена основным вопросам эффективности вычислений в её современном понимании.

Чтобы количественно охарактеризовать эффективность вычислений, понадобится ввести несколько определений. Под *вычислительной задачей* будем понимать утверждение о том, *что* требуется вычислить. Например, задача сортировки состоит в том, чтобы найти такую перестановку заданного конечного набора чисел $x_1, x_2, ..., x_N$, чтобы в этой перестановке каждое следующее число оказалось не меньше предыдущего. Или, например, задача квадратичной оптимизации состоит в том, что необходимо вычислить вектор $\mathbf{x}$ такой, что квадратичная форма

$$\frac{1}{2}(\mathbf{x}, H\mathbf{x}) + (\mathbf{x}, \mathbf{c}) \tag{1.1}$$

принимает наименьшее возможное значение для известной матрицы H и известного вектора $\mathbf{c}$ [4]. Или, например, задача может состоять в вычислении определителя известной матрицы $\det A$.

Понятие *алгоритм* интуитивно воспринимается как последовательность действий для некоторой вычислительной машины. Подразумевается, что добросовестно выполнив эту последовательность электронная вычислительная машина предоставит нам финальный результат работы алгоритма — решение той или иной вычислительной задачи. Иными словами, алгоритм описывает, *как* нужно вычислять для достижения результата.

В случае задачи сортировки мы могли бы поступить разными способами. Во-первых, перебирать все перестановки и проверять каждую на выполнение условия. Во-вторых, использовать один из множества известных алгоритмов сортировки (Кнут 2019c).

Для задачи квадратичной оптимизации можно было бы вычислить требуемый вектор, используя аналитическую формулу

$$\mathbf{x} = -H^{-1}\mathbf{c}, \tag{1.2}$$

либо вычислять вектор $\mathbf{x}$ итеративно (см., например, раздел 4.2.2). Для достижения цели в первом случае нужно выбрать алгоритм обращения матрицы и алгоритм умножения матрицы на вектор.

Для вычисления определителя матрицы можно было бы воспользоваться определением:

$$\det A \equiv \sum_{\alpha_1,\alpha_2,\ldots,\alpha_n} (-1)^{N(\alpha_1,\alpha_2,\ldots,\alpha_n)} \cdot a_{1\alpha_1} a_{2\alpha_2} \ldots a_{n\alpha_n}, \tag{1.3}$$

либо применить алгоритм поиска собственных значений λ_i матрицы A и затем их перемножить:

$$\det A = \lambda_1 \cdot \lambda_2 \cdot \ldots \cdot \lambda_N \tag{1.4}$$

[3] Кроме того, проводить расчёты эффективно считается хорошим тоном в среде компьютерных наук.

[4] В данной главе мы не рассматриваем вопрос *зачем* могло бы потребоваться вычислять минимизирующий такую квадратичную форму вектор, но подобным вопросам посвящено большинство остальных глав пособия.

Для решения одной и той же задачи можно предложить несколько алгоритмов, позволяющих получить эквивалентные ответы (либо в точности одинаковые, либо близкие друг к другу в пределах требуемой точности вычислений), а значит, предстоит разобраться, как сравнивать их эффективность между собой.

Реализация алгоритма[5] — конкретный исходный код программы на некотором языке программирования, написанный определённым автором. В качестве примеров реализаций алгоритмов можно назвать популярные пакеты для языка программирования Python, такие как `numpy`, `scipy`, `scikit-learn` и другие. Эти пакеты содержат готовые функции для операций с матрицами и векторами, численной оптимизации, методов машинного обучения и т.д.

Другим примером реализаций алгоритмов служит, например, код домашних заданий, выполняемых студентами в рамках курса «Обработка и анализ данных физического эксперимента» на Факультете физики ВШЭ. Реализации одного и того же алгоритма могут существенно отличаться по скорости работы, как за счёт используемого языка программирования или оптимизирующего компилятора, так и за счёт квалификации программиста.

Пожалуй, самый наивный способ сравнения эффективности вычислений состоит в том, чтобы замерить время работы реализаций различных алгоритмов на конкретной ЭВМ, используя секундомер[6]. Хотя такой способ и применяется на практике, его одного недостаточно, так как он обладает рядом недостатков. Во-первых, сравниваются реализации, а значит, нужно потрудиться реализовать несколько алгоритмов, запустить реализации и несколько раз получить *эквивалентные* ответы. Ради того, чтобы просто получить ответ, было бы достаточно иметь одну реализацию, и время вычислений для неё было бы заведомо меньше, чем время вычислений для всего набора.

Во-вторых, невозможно заранее предсказать, как изменится время работы при изменении количества входных данных алгоритма или каких-то иных его параметров, например, точности вычисления результата. Потребуется запустить реализацию с новым набором данных, и даже если мы дождёмся завершения работы, то весь смысл в измерении эффективности опять теряется. Это наводит на мысль, что необходим способ оценки эффективности алгоритма до его реализации и запуска. И такой способ, к счастью, есть, но сначала необходимо уяснить отличия алгоритма от любой другой последовательности действий, например, от кулинарного рецепта.

1.1 Алгоритм как математический объект

Исторически первым, вероятно, формализовал понятие алгоритма британский математик Алан Тюринг в 1930-х годах (Turing 1937). Мы же обратим-

[5] implementation

[6] Часто такую меру называют *wall-clock time*, т.е. время, которое показывают обычные часы (например, настенные).

ся к определению, закреплённому американским математиком Дональдом Кнутом в своём фундаментальном многотомнике «Искусство программирования» (Кнут 2019a).

Рассмотрим следующую четвёрку $(Q, I \in Q, \Omega \in Q, f : Q \to Q)$, здесь Q, I, Ω — некоторые абстрактные множества.

- Q — множество всех состояний алгоритма. Попросту говоря, если мы рассмотрим набор всех используемых алгоритмом «переменных», то все возможные комбинации их значений и определяет нам множество Q.
- I — начальное подмножество, это состояния, из которых алгоритм может стартовать. Оно представляет собой способ задания каких-то входных данных или параметров.
- Ω — подмножество конечных состояний.
- Функция f переводит состояние алгоритма само в себя до тех пор, пока алгоритм не завершится.

Мы считаем, что $f(q) = q$ для $\forall q \in \Omega$, а последовательность состояний $q_{k+1} = f(q_k)$ в конечном итоге сходится к некоторому $q \in \Omega$. В этот момент говорят, что исполнение алгоритма завершено.

Может показаться непонятным, почему бы не задать функцию f таким образом, чтобы все последовательности сходились за один шаг. Подразумевается, что функция f может выполнять только элементарные преобразования. Понятно, что теория должна адекватно описывать реальный моделируемый объект, поэтому в качестве таких элементарных преобразований выбираются те операции, которые центральный процессор умеет выполнять аппаратно, например:

- загрузка значения из памяти в регистр (чтение памяти),
- сравнение значений в регистрах,
- арифметические операции над регистрами (сложение, вычитание, умножение, деление),
- выгрузка значения из регистра в память (запись памяти).

Мы исходим из того, что выполнение элементарной операции каждого вида всегда занимает одинаковое время. Без замены оборудования такие операции сами по себе нельзя улучшить, т.е. сделать ещё более быстрыми.

Итак, алгоритм — это математический объект, способный решить для нас некоторую задачу по преобразованию данных. Фактически, алгоритм — это ещё один способ задать некоторое отображение (функцию) $F : \mathbf{X} \to \mathbf{Y}$, наряду с другими способами, такими как:

- записать функцию в виде аналитического выражения (например, $F(x) \equiv x + x^2$);

- записать таблицу, если множества $\mathbf{X}$ и $\mathbf{Y}$ конечны;

- обозначить бесконечный ряд, интеграл с параметром, решение дифференциального (или алгебраического) уравнения новой функцией.

Пожалуй, наиболее показательно алгоритмы в роли математических функций проявят себя в разделе 8.1.

Все эти способы не эквивалентны с точки зрения практического удобства доведения вычислений до числа, к счастью, в ряде случаев можно строго доказать эквивалентность определений, данных разными способами. Это позволяет в различных ситуациях подменять одно определение другим, исходя из практических требований. Примером является функция Бесселя $J_\nu(x)$, которую можно определить и как ряд, и как интеграл с параметром, и как одно из решений дифференциального уравнения, но все эти определения приведут к одной и той же функции. Аналогично этому, можно формально доказать корректность алгоритма, т.е. строго убедиться, что алгоритм действительно решает заявленную задачу.

Теперь алгоритм, как и любой другой математический объект, может быть подвергнут анализу и исследованию свойств. Такими свойствами могут быть, например, скорость работы (длина последовательности q_k, т.е. фактически число затраченных элементарных операций), количество требуемой оперативной памяти или точность вычисления результата.

1.2 O-нотация

На практике интересным вопросом является зависимость числа затраченных элементарных операций или количество требуемой дополнительной памяти от размера входных данных или необходимой точности вычислений. Зависимость объёма работы от размера входных данных часто называют *вычислительной сложностью*[7].

Чаще всего входные данные представлены в виде набора элементов одинаковой природы. Например, набор чисел для сортировки или матрица для обращения. В таком случае под размером входных данных естественно понимать число таких элементов: количество чисел для сортировки, либо число строк квадратной матрицы, соответственно.

Оказывается, для многих алгоритмов можно подсчитать число операций аналитически, однако здесь есть две сложности. Во-первых, число операций может зависеть не только он размера входных данных, но и от их значения. Например, если мы хотим отсортировать уже упорядоченный набор чисел с помощью алгоритма сортировки вставками (Кнут 2019c), то можно показать, что такая процедура займёт меньше времени, чем в случае сортировки неупорядоченного набора. Должны ли мы изучать скорость работы алгоритма для каждого допустимого набора входных параметров? Такая

[7] computational complexity

детализация была бы излишней, поэтому на практике изучают число операций в *среднем случае*[8], когда оно усредняется по ансамблю всех возможных входных параметров, либо число операций в *худшем случае*[9], когда оно изучается для такого набора входных данных, с которым алгоритм будет работать дольше всего.

Во-вторых, точное аналитическое выражение, если его можно получить, как правило, выглядит громоздко и содержит в себе различные неэлементарные функции (например, гармоническое число $H(N) \equiv \sum_{j=1}^{N} 1/j$). К счастью, оказывается, что достаточно рассматривать лишь асимптотическое поведение зависимости требуемых элементарных операций (либо количества требуемой дополнительной памяти) от размера входных данных при стремлении этого размера к бесконечности.

Напомним, что из курса математического анализа известны следующие определения:

- $f(x) = O(g(x))$ значит, что

$$\exists x_0, C > 0 : \quad \forall x \geq x_0 \quad |f(x)| \leq C|g(x)|$$

- $f(x) = \Theta(g(x))$ значит, что

$$\exists x_0, C_1 > 0, C_2 > 0 : \quad \forall x \geq x_0 \quad C_1|g(x)| \leq |f(x)| \leq C_2|g(x)|$$

Т.е. если $f(x) = O(x^2)$, то из этого следует, что, например, $f(x) = O(x^4)$. Ирония состоит в том, что в мире компьютерных наук по сложившейся традиции используют обозначение $O(g(x))$ в смысле $\Theta(g(x))$. Вероятно, это обусловлено тем, что символ Θ был менее распространён на первых компьютерах. Но на практике никакой путаницы из-за этого не возникает. Например, говорят, что алгоритм сортировки слияниями (Кнут 2019c) имеет вычислительную сложность $O(N \log N)$ в худшем случае, а «пузырьковый» алгоритм сортировки имеет вычислительную сложность $O(N^2)$ в худшем случае. При этом подразумевается, что для алгоритмов сортировки подсчитываются операции попарного сравнения чисел. Или, например, умножение двух квадратных матриц размера N с помощью наивного алгоритма потребует $O(N^3)$ операций умножения (и в среднем и в худшем случае, поэтому уточнение опускается). Обратим внимание, что основание логарифма не ставится, так как не имеет смысла под знаком асимптотики.

Аналогично этому, с помощью O-нотации можно оценить количество требуемой дополнительной памяти. Например, говорят, что операция поиска максимального значения в наборе чисел требует $O(1)$ дополнительной памяти, т.е. размер дополнительной памяти не зависит от размера входных данных. Однако, операций сравнения всё равно необходимо $O(N)$.

Итак, O-нотация — удобный формализм для изучения вычислительной сложности алгоритмов. Во-первых, асимптотическая вычислительная

[8] average-case
[9] worst-cast

сложность имеет простой вид: на практике большинство алгоритмов имеют один из следующих вариантов вычислительной сложности — $O(1)$, $O(\log N)$, $O(N)$, $O(N \log N)$, $O(N^p)$.

Во-вторых, асимптотика является хорошим начальным приближением, чтобы судить об эффективности алгоритма. Считается, что чем медленнее асимптотика, тем эффективнее алгоритм. Поэтому, например, говорят, что сортировка слияниями лучше, чем «пузырьковый» алгоритм сортировки, так как функция $N \log N$ растёт медленнее, чем N^2 при $N \to \infty$.

В-третьих, асимптотическая вычислительная сложность, дополненная методом прямого измерения вычислительного времени, позволяет хорошо экстраполировать, т.е. отвечать на вопросы вида «Что случится, когда данных станет в два раза больше?».

Впрочем, стоит помнить, что, согласно формальному определению $N \to \infty$, при этом какой именно N можно считать достаточно большим, чтобы стать бесконечностью — вопрос сложный. Иначе говоря, O-нотация не учитывает мультипликативную константу. Таким образом лучшая асимптотика не всегда означает лучший алгоритм, классическим примером является задача перемножения двух квадратных матриц размера N.

Рассмотрим три следующих подхода. Во-первых, наивный подход, состоящий в прямом использовании определения матричного умножения

$$c_{ik} = \sum_j a_{ij} b_{jk}, \tag{1.5}$$

легко проверить, что в этом случае потребуется $O(N^3)$ операций умножения.

Во-вторых, алгоритм Штрассена (Strassen 1969), в котором задействовано $O(N^{\log_2 7}) = O(N^{2.8})$ операций умножения. Казалось бы, что такой алгоритм лучше, но на практике он превосходит наивный подход только при N больше сотен или тысяч. Тут надо оговориться, что число это очень примерное и должно быть замерено для каждой конкретной реализации и модели процессора.

В-третьих, алгоритм Копперсмита-Винограда (Coppersmith и Winograd 1990), в котором требуется $O(N^{2.375})$ операций умножения, но практический смысл такого алгоритма сомнителен: пороговое значение N, при котором он опережает алгоритм Штрассена, оказывается очень велико. Судя по всему, идёт о матрицах с числом строк больше миллиона, а возможно, существенно больше.

Вообще говоря, существуют и обратные примеры. Так, например, для

решения задачи линейного программирования [10]:

$$\max \sum_{j=1}^{N} p_j x_j, \tag{1.6}$$

$$\sum_{j=1}^{N} a_{ij} x_j \le b_i, \tag{1.7}$$

где x_j — неизвестные действительные числа, а a_{ij}, b_i, p_j суть известные действительные числа, применяется симплекс-метод (предложенный Джорджем Данцигом в 1940-х), который имеет экспоненциальную сложность в худшем случае. Пример худшего случая известен под названием куба Клее-Минти. Однако, в среднем этот алгоритм требует линейного числа шагов и является одним из признанных эффективных способов решения указанной задачи.

И теперь про примеры из начала главы стало всё ясно. Для решения задачи сортировки следует предпочесть специализированный алгоритм, вычислительная сложность которого $O(N \log N)$, наивному перебору всех перестановок, сложность которого $O(N!)$. Оба предложенных метода решения задачи квадратичной оптимизации имеют сложность $O(N^3)$. Для вычисления определителя матрицы нужно воспользоваться разложением на собственные значения, со сложностью $O(N^3)$, а не наивным суммированием по всем перестановкам, количество которых опять $O(N!)$.

1.3 Теория алгоритмов

Мы видели, что для некоторых задач существует несколько алгоритмов решения, а исследование асимптотической вычислительной сложности даёт оценку их эффективности. Однако, может возникнуть вопрос, можно ли для некоторой задачи придумать новый алгоритм, который будет лучше, чем известные алгоритмы в смысле асимптотической вычислительной сложности? Оказывается, для некоторых задач можно сформулировать ответ на этот вопрос в виде соответствующих теорем.

Так, например, можно доказать теорему, что любой алгоритм сортировки, основанный на попарных сравнениях, требует не менее чем $\lceil \log N! \rceil$ операций сравнения. Иначе говоря, вычислительная сложность указанных алгоритмов сортировки — $\Omega(N \log N)$ (Кнут 2019c).

Запись $f(x) = \Omega(g(x))$ значит, что

$$\exists x_0, C > 0 : \quad \forall x \ge x_0 \quad |f(x)| \ge C|g(x)|.$$

Отметим, что по сложившейся традиции вместо обозначения $\Omega(g(x))$ часто используется обозначение $O(g(x))$, но, как и прежде, путаницы не возникает.

[10] Напомним, что термин «задача математического программирования» обозначает задачу оптимизации и в данном случае не имеет отношения к программированию ЭВМ.

Заметим, что теорема ничего не говорит о специализированных алгоритмах сортировки. Так, например, для целых положительных чисел существует алгоритм поразрядной сортировки, асимптотическая сложность которого $O(N\omega)$, где ω — размер числа в битах (Кнут 2019c).

Напротив, наилучшая сложность матричного умножения точно не известна, но предполагается, что она составляет $\Omega(N^{2+\epsilon})$, где $0 \leq \epsilon < 0.372$, причём верхняя граница уменьшается по мере изобретения новых алгоритмов.

Кроме того важно, что при сужении задачи, т.е. рассмотрения более частного случая, как правило, становится возможно предложить более эффективный алгоритм с точки зрения асимптотической вычислительной сложности. Например, рассмотрим обращение квадратной матрицы размером N. Хотя теоретическая сложность обращения произвольной матрицы такая же, как у умножения квадратных матриц (см., например, Кормен и др. 2019), сложность популярных алгоритмов обращения составляет $O(N^3)$. Но если мы рассматриваем верхнюю треугольную матрицу или нижнюю треугольную матрицу, то можно предложить алгоритм, имеющий вычислительную сложность $O(N^2)$. Если рассматривается трёхдиагональная матрица, то вычислительная сложность такой задачи уже $\Omega(N)$.

1.3.1 NP-задачи

Теперь мы поняли, что для некоторых задач мы можем теоретически оценить вычислительную сложность самого эффективного алгоритма. Иначе говоря, можно доказать, что получить более эффективный алгоритм без переформулировки задачи невозможно. Возникает вопрос, а бывают ли такие задачи, сложность которых настолько велика, что ставит под сомнение практическую ценность применения ЭВМ для их решения? Оказывается, что существует целый класс задач, которые невозможно без переформулировки эффективно решать на существующих вычислительных системах.

Примером является следующая задача дискретной оптимизации, известная как «задача о рюкзаке»:

$$\max \sum_{j=1}^{N} p_j x_j, \tag{1.8}$$

$$\sum_{j=1}^{N} \omega_j x_j \leq C, \tag{1.9}$$

где неизвестные переменные $x_j \in \{0, 1\}$, а параметры ω_j, C и p_j суть известные действительные числа.

По легенде, злоумышленник проникает в музей и видит там N экспонатов, каждый стоимостью p_j и весом ω_j. Задача грабителя — унести ценностей на наибольшую сумму, но грузоподъёмность его рюкзака составляет C и при её превышении рюкзак порвётся, поэтому для каждого экспоната нужно принять решение x_j — оставить ли экспонат в музее или положить в

рюкзак и унести. Основная сложность этой задачи заключается в том, что каждый экспонат нужно либо взять целиком, либо целиком оставить.

Другим примером таких задач является задача целочисленного программирования:

$$\max \sum_{j=1}^{N} c_j x_j, \tag{1.10}$$

$$\sum_{j=1}^{N} a_{ij} x_j \leq b_i, \tag{1.11}$$

где $x_i \in \mathbf{Z}$, а a_{ij}, b_i и c_j суть известные целые числа.

Рассмотрим на примере задачи о рюкзаке связанную с проблемой терминологию. Обычно, помимо основной задачи оптимизации, рассматривают две сопряжённые с ней задачи. Во-первых, *задача разрешимости*: пусть задано t, существует ли набор x_j такой, что $\sum_{j=1}^{N} p_j x_j \geq t$, при условии (1.9)? Понятно, что если для задачи разрешимости существует алгоритм решения, то ответ к изначальной задаче может быть найден с помощью деления отрезка пополам. Если существует алгоритм с асимптотической вычислительной сложностью $O(N^p)$, то говорят, что задача принадлежит классу P.

Во-вторых, *задача верификации*: пусть задано t и набор x_j, нужно проверить, являются ли они решением задачи $\sum_{j=1}^{N} p_j x_j \geq t$, при условии (1.9). Если существует алгоритм, позволяющий выполнить проверку за $O(N^p)$, то говорят, что задача принадлежит классу NP. Для задачи о рюкзаке такой алгоритм, очевидно, существует.

NP-полными (NP-complete) называются такие задачи, к которым может быть сведена любая NP задача за полиномиальное время. Класс P содержится внутри NP, но совпадают ли они полностью — непонятно. Этот вопрос является одной из нескольких «математических задач тысячелетия» по версии института Клея[11]. Если бы классы полностью совпали ($P = NP$), тогда наличие полиномиального алгоритма для задачи верификации гарантировало бы нам, что где-то существует полиномиальный алгоритм для решения задачи разрешимости. А вот если классы не совпадают ($P \neq NP$), это значит, что существуют задачи, для которых не может существовать полиномиального алгоритма для задачи разрешимости.

На практике NP-полные задачи переформулируют в вид, допускающий существование алгоритмов с полиномиальной вычислительной сложностью. Например, вместо решения «задачи о рюкзаке» в исходном виде, можно найти такой набор $x_j \in \{0, 1\}$, что $\sum_{j=1}^{N} p_j x_j \geq \alpha \max \sum_{j=1}^{N} p_j x_j$, при условии (1.9). Т.е., например, при $\alpha = 0.95$ будет найдено решение, которое хуже оптимального не более чем на 5%.

[11] `https://www.claymath.org/millennium-problems`

1.4 Структуры данных

Итак, теперь нам известно, как измеряется эффективность различных алгоритмов и сложность вычислительных задач. В первом случае это позволяет нам сравнивать алгоритмы и выбирать наиболее подходящий, а во втором случае иметь ориентир, насколько выбранный алгоритм можно улучшить.

Пока в стороне оставался вопрос, откуда в принципе берутся новые алгоритмы. Понятно, что в конечном счёте это творческий процесс, однако, хорошо было бы иметь набор удачных универсальных заготовок.

При обработке данных часто бывает так, что сложности возникают уже на этапе их подготовки, до применения каких-то математизированных методов анализа. Дело может быть в том, что в рамках экспериментов данные записываются в неудобном порядке или для анализа требуется свести воедино измерения, выполненные в рамках разных экспериментов.

Например, рассмотрим два астрономических набора данных (или «каталога», как говорят астрономы). Набор данных космического эксперимента GAIA (Gaia Collaboration и др. 2023) содержит примерно $N_1 = 1.5 \cdot 10^9$ уникальных записей о точных положениях и расстояниях до звёзд Галактики. Набор данных наземного обзора ZTF (Bellm и др. 2018) содержит примерно $N_2 = 2.7 \cdot 10^9$ уникальных фотометрических кривых блеска (зависимостей яркости звезды от времени), но не содержит никакой информации о расстоянии, так как в рамках этого эксперимента его невозможно было определить. Сразу же возникает желание объединить эти два набора данных, чтобы в одном наборе анализировать как расстояние до звезды, так и её яркость. Трудность, с которой мы сталкиваемся, состоит в том, что положения звезд на небе измеряются с некоторой погрешностью как в первом, так и во втором случае, а это значит, что нам нужно для каждой записи из первого набора данных найти *ближайшего соседа* из второго набора данных. Строгое сравнение координат, записанных в виде пары чисел, здесь не сработает как раз из-за погрешности измерений.

Оценим, можно ли реализовать это объединение наивным способом: для каждой из записей первого набора данных рассчитаем расстояния до каждой из записей второго набора данных и попутно найдём минимальное, которое и будет указывать нам на запись, образующую нужную нам связь. Всего таких пар расстояний будет порядка $N_1 N_2 \approx 10^{18}$. Предположим, что координаты звёзд в каждом наборе данных представлены в виде пары чисел с плавающей точкой двойной точности, т.е. суммарно 16 байт. Координаты из двух наборов данных займут примерно $16\,(N_1 + N_2) = 16 \cdot (1.5 + 2.7) \cdot 10^9 \approx$ 60GB памяти. В настоящее время компьютеры с 60GB и более оперативной памяти совсем не редки, но для интереса рассмотрим два случая: когда данные хранятся на хорошем быстром твердотельном жёстком диске и когда данные предварительно загружены в более быструю оперативную память. Для наивного подхода, состоящего в подсчёте всех попарных расстояний, оценим суммарное количество информации, которое необходимо прочитать и пропустить через центральный процессор: результат будет $\approx 2 \cdot 16 N_1 N_2 = 2 \cdot 16 \cdot 1.5 \cdot 2.7 \cdot 10^{18} = 120 \cdot 10^9$GB.

Пусть чтение с очень хорошего твердотельного жёсткого диска занимает около $2\,\mathrm{GB\,s^{-1}}$, тогда время, затраченное на чтение данных, займёт примерно 1900 лет. Если данные уместились в оперативную память, то можно обеспечить на порядок большую скорость, возьмём в качестве оценки $40\,\mathrm{GB\,s^{-1}}$ и получим приблизительно 100 лет. Отметим, что обе оценки оптимистичные и не учитывают дополнительные накладные расходы в виде задержек с доступом к памяти. Понятно, что эти оценки скорости работы говорят о том, что наивный подход оказывается непрактичным и нужно искать альтернативный.

Из приведённого примера видно, что основная проблема возникает из-за того, что требуется $O(N_1 N_2)$ операций чтения данных. Оказывается, что это количество можно снизить, расположив данные в памяти удобным способом.

Структуры данных — способы расположения информации в адресном пространстве оперативной памяти или жёсткого диска, которые помогают нам составлять более эффективные алгоритмы. Примерами основных популярных структур данных являются, например: вектор (массив), список, деревья, хеш-таблицы, которые будут по очереди рассмотрены далее. Речь идёт о наборах (коллекциях) элементов одинаковой природы. В примере выше мы работали с двумя наборами, элементами первого были пара координат и расстояние, а элементами второго — пара координат и таблица зависимости яркости от времени на некоторой сетке.

Перед тем как познакомиться с какой-нибудь из структур данных, следует вспомнить, что из себя представляет операция чтения (и записи) памяти. На самом деле, с точки зрения стороннего наблюдателя, микросхема памяти работает предельно просто [12]: она меняет одно целое число, которое принято называть *адресом*, на другое число, подобно тому как в гардеробе меняют номерок на одежду. Не вдаваясь в детали, представим, что оба числа закодированы некоторым числом бит, например, 32. Тогда адрес подаётся на входные контакты микросхемы так, что каждому из битов соответствует один физический контакт микросхемы. Второе число — это и есть хранимые данные: как только адрес подан на вход микросхемы, на каждом из выходных контактов возникает логический сигнал, соответствующий значению бита. Будем считать, что скорость работы микросхемы не зависит от значения адреса, тогда говорят, что сложность произвольного доступа к памяти составляет $O(1)$ [13]. Запись производится аналогичным образом, только теперь все контакты играют роль входа.

Волшебство заключается в том, что, во-первых, адреса тоже можно хранить в этой же микросхеме в роли данных, а во-вторых, процессор умеет

[12] На самом-то деле, доступ к памяти в современном устройстве с многоядерным процессором работает предельно сложно. Стоит лишь упомянуть о многоуровневом кэше процессора, многоканальных контроллерах памяти и прямом доступе к памяти со стороны периферийного оборудования. К счастью, для понимания структур данных всё это не важно, и мы удовлетворимся простой моделью происходящего.

[13] Примером устройства, где время доступа к данным зависит от адреса, является накопитель с использованием магнитной ленты, но таковые ныне используются лишь для холодного хранения данных.

выполнять арифметические операции над адресами. С формальной точки зрения, мы имеем дело с некоторым непрерывным *адресным пространством*, примерами которого являются оперативная память, жёсткий диск или один файл в рамках диска. Структуры данных описывают правила размещения коллекции однотипных данных в таком адресном пространстве, и, следовательно, задают алгоритмы элементарных операций над этими коллекциями. Такими операциями, например, являются:

- поиск элемента по значению,
- добавление нового элемента в коллекцию,
- удаление старого элемента из коллекции,
- замена значения элемента.

Понятно, что для каждой структуры данных алгоритмы операций коллекции будут разными, значит знание сильных и слабых сторон каждой из структур данных позволяет эффективно комбинировать их в рамках более общих алгоритмов и программ. Например, если бы мы смогли разместить данные из каталога ZTF в такую структуру данных, которая предусматривала бы операцию поиска ближайшего соседа произвольной точки со сложностью $O(\log N_2)$, то мы смогли бы решить нашу исходную задачу за $O(N_1 \log N_2)$ чтений данных, т.е. потратили бы на неё несколько минут.

Рассмотрим основные структуры данных.

1.4.1 Массив

Массив (известен также под названием *вектор*, а в языке программирования Python — список или `numpy.array` из пакета `numpy`) — одна из самых простых структур данных, поэтому знакомство со структурами данных разумно начать с него. Пусть мы хотим разместить некоторый набор X из N объектов, требующих s байт памяти для каждого. Например, рассмотрим пары чисел с плавающей точкой из предыдущего примера. Разместим их в памяти последовательно, без пропусков, в том порядке, в котором они нам предоставлены.

Нам достаточно запомнить адрес нулевого элемента (который мы назовём адресом массива) a_0, чтобы получить доступ к любому из элементов, зная индекс i этого элемента. Адрес a_i элемента номер i можно легко вычислить следующим образом:

$$a_i = a_0 + i \cdot s. \tag{1.12}$$

Напомним, что произвольный доступ к любому адресу памяти имеет константную сложность, значит *чтение или запись i-го элемента массива* имеет сложность $O(1)$. Допустим, мы хотим проверить, содержится ли элемент со значением x' в нашем массиве. В этом случае лучшим вариантом оказывается последовательное считывания всех элементов $x \in X$ и сопоставление

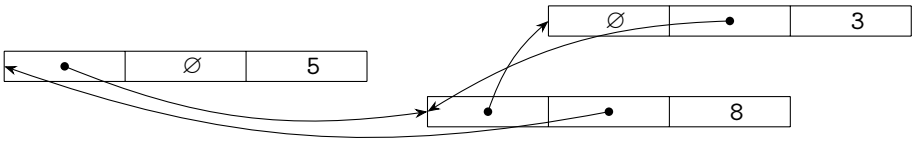

Рис. 1.1: Схематичное изображение двусвязного списка, хранящего последовательность $\{5, 8, 3\}$. Прямоугольники обозначают области адресного пространства. Стрелки показывают, что в заданном месте записан адрес соответствующего блока. Символ $\varnothing$ обозначает пустой адрес.

с образцом x', в худшем случае, когда элемент x' отсутствует в массиве, это займёт $O(N)$ операций чтения и сравнения. Тогда говорят, что *поиск по значению в массиве* имеет сложность $O(N)$. Операции вставки или удаления элемента из позиции i тоже имеют сложность $O(N)$, это связано с тем, что при добавлении нужно освободить i-ую позицию (т.е. переписать все значения на соседнее положение вправо), а при удалении нужно переместить на освободившуюся следующий элемент и т.д. Отметим, что добавление нового элемента перед нулевым элементом невозможно по соглашению, а добавление элемента после последнего тоже имеет сложность $O(N)$, это уже связано с техническими ограничениями: адресное пространство (память) перед первым элементом или после последнего элемента может быть уже занято хранением не имеющих отношения к массиву данных. Это значит, что для добавления нового элемента в конец массива потребуется переместить весь массив в новое, более просторное место.

Заметим, что если мы отказываемся от необходимости хранить элементы в некотором заданном порядке, мы могли бы отсортировать такой массив, и тогда *поиск по значению в отсортированном массиве* можно осуществлять за $O(\log N)$ операций, используя метод *двоичного поиска*[14].

1.4.2 Список

Список (в языке программирования Python — `collections.deque`) может быть односвязным или двусвязным. Общий принцип состоит в том, что для каждого из N объектов в адресном пространстве выбирается своё индивидуальное место, достаточное для хранения этого объекта и адреса, указывающего, где хранится следующий по порядку элемент списка (такой список называется *односвязным*), либо пары адресов, указывающих на предыдущий и следующий элементы (такой список называется *двусвязным*). Адрес a_0, указывающий на место хранения нулевого элемента, и есть адрес списка. Схематично список показан на рис. 1.1.

Отметим различия между списком и массивом. Во-первых, списку требуется больше памяти для хранения тех же самых элементов. Во-вторых, чтение или запись i-го элемента потребуют i последовательных чтений, т.е.

[14] binary search

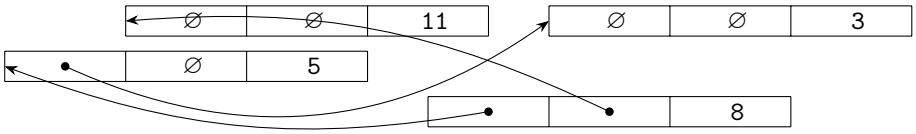

Рис. 1.2: Схематичное изображение двоичного дерева, корню которого приписано число 8. Прямоугольники обозначают области адресного пространства. Стрелки показывают, что в заданном месте записан адрес соответствующего блока. Символ ∅ обозначает пустой адрес.

чтение или запись i-го элемента списка имеет сложность $O(N)$. Чтобы узнать, где лежит элемент номер i, мы должны прочитать элемент номер $i-1$, где записан нужный нам адрес, и т.д. Заметим, что и *поиск по значению в списке* имеет сложность $O(N)$.

Однако, тут скрывается важное преимущество двусвязного списка: если мы нашли интересный нам элемент x', который располагается по адресу a', то удаление этого элемента, а так же вставка нового элемента до или после него, займёт $O(1)$ операций, так как потребует только исправления адресов в смежных элементах. Это же утверждение верно для вставки и удаления в конец и начало списка. Для односвязного списка сложность $O(1)$ имеют только некоторые из этих операций.

Списки используются, например, для представления в памяти разреженных матриц и векторов, т.е. таких, большинство элементов которых равно 0, или для организации очередей, когда необходимо часто удалять элементы из начала списка и добавлять новые элементы в конец.

1.4.3 Деревья поиска

Дерево — это математическая абстракция, частный случай графа. Деревом называется односвязный ациклический граф, одна из *вершин* (узлов) которого назначена *корнем* (зафиксирована). Как только мы фиксируем корень, некоторые вершины дерева становятся *листьями*, т.е. тупиковыми вершинами при обходе дерева, начиная от корня. Для каждого из листьев можно подсчитать число вершин на пути от корня до этого листа, назовём такое число расстоянием или *глубиной (высотой) листа*. Максимальную глубину листа среди всех листьев назовём *глубиной (высотой) дерева*.

Особый практический интерес в данном разделе для нас будут представлять *двоичные (бинарные) деревья*: у такого дерева каждый узел имеет не более двух потомков. Практическая польза связана с тем, что двоичные деревья достаточно просто представлять в линейном адресном пространстве: по аналогии со списком каждый узел будет соответствовать области памяти, где хранятся данные, приписанные к этому узлу, и дополнительно два адреса, указывающие на поддеревья (их для удобства называют левым и правым поддеревьями). Адресом дерева будем считать адрес его корня. Таким образом, зная адрес дерева, можно за конечное число операций по-

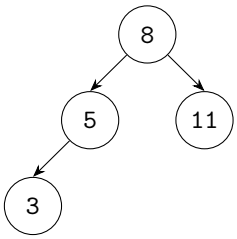

Рис. 1.3: Схематичное изображение двоичного дерева поиска. Возможное расположение элементов в адресном пространстве показано на рис. 1.2.

лучить доступ к данным, приписанным к любому из его узлов. Схематично двоичное дерево показано на рис. 1.2.

Следует сразу сказать, что деревья оказались настолько эффективным инструментом, что применяются не только для организации коллекций, но и во многих алгоритмах машинного обучения (см., например, раздел 8.1). Одним из часто используемых деревьев — *двоичное дерево поиска*. Двоичное дерево поиска устроено следующий образом:

- каждый элемент x коллекции X приписывается своему узлу дерева;
- все элементы x коллекции X попарно сравнимы между собой, иначе говоря, для них определена операция «меньше» ($<$);
- для каждого промежуточного узла хранимое (приписанное) ему значение x таково, что $x_L < x$, где x_L — любое значение из левого поддерева (приписанное любому из узлов левого поддерева), и, одновременно, $x < x_R$, где x_R — любое значение из правого поддерева.

Пример двоичного дерева поиска представлен на рис. 1.3.

Конечно, немного расстраивает необходимость уметь сравнивать элементы. Если для чисел операция сравнения вводится естественным образом, а для строк её очевидно можно определить, чтобы сохранялся лексографический порядок (т.е. алфавитный), то для векторов или комплексных чисел ситуация непонятная. Тем не менее, рассмотрим, какие операции и как можно выполнять над двоичным деревом поиска.

Из последнего свойства двоичного дерева поиска видно, что, в отличие от массива и списка, теперь каждый элемент не имеет своей произвольной позиции, а вернее, его положение полностью зависит от значения этого элемента. В массиве и списке мы могли бы произвольно менять элементы местами, т.е. сохранять нашу коллекцию в произвольном порядке. В дереве поиска любая коллекция хранится в определённом порядке, навязанном структурой данных. Это значит, что операция чтения или записи i-го элемента не существует для дерева поиска, так как не имеет смысла.

Зато это свойство позволяет сформулировать следующий алгоритм *поиска элемента x' по значению в двоичном дереве поиска*. Начнём с корня,

т.е. мы знаем значение, приписанное корню x, и адреса узлов корней левого и правого поддеревьев.

- Если вдруг оказалось, что $x' = x$, то элемент найден.
- Если $x' < x$, тогда, согласно свойству дерева поиска, узла со значением x' точно нет в правом поддереве. Если левое поддерево не пусто, то элемента в нем тоже может не быть, что мы проверим, применив этот же алгоритм поиска элемента x' к левому поддереву, адрес корня которого нам уже известен.
- Если $x < x'$, тогда, согласно свойству дерева поиска, узла со значением x' точно нет в левом поддереве, а вот в правом поддереве имеет смысл его поискать, применив этот же алгоритм.

Итак, либо узел, соответствующий элементу x', будет обнаружен на каком-то этапе, либо будет обнаружен лист дерева. В последнем случае можно строго говорить об отсутствии элемента x' в коллекции, так как по построению дерева поиска такой элемент обязан был бы оказаться в левом или правом поддереве листа. Понятно, что сложность такого алгоритма пропорциональна числу сравнений, т.е. глубине финального узла. В худшем случае, когда элемент отсутствует, сложность алгоритма поиска составит $O(h)$, где h глубина дерева.

Вставка нового элемента осуществляется аналогичным образом: выполняется неудачный поиск и дописывание нового листа в качестве потомка финального листа. Сложность такого алгоритма опять пропорциональна числу сравнений, так как сама вставка нового листа потребует лишь конечного числа записей адресов в соответствующих узлах. Операция *удаления элемента* чуть более замысловата, но тоже начинается с поиска по значению элемента, который требуется удалить. Некоторую сложность представляет случай, когда у удаляемого узла два потомка, но остальные случаи тривиальны, и операция удаления опять имеет сложность $O(h)$.

Сбалансированные деревья

Было бы здорово, если бы оказалось, что глубина дерева h пропорциональна $\log N$, где N — число узлов дерева, и, следовательно, элементов коллекции. В общем случае так, конечно, сделать нельзя, но для *сбалансированного дерева* такое требование выполняется, что позволяет нам осуществлять операцию поиска элемента за $O(\log N)$. Сбалансированным двоичным деревом называется такое двоичное дерево, у которого глубины левого и правого поддеревьев для каждого узла различаются не более чем на единицу [15]. На

[15] Иногда различают *сбалансированное дерево* и *идеально сбалансированное дерево*, у которого полностью заполнены все уровни, кроме последнего, а значит, глубины всех листьев различаются не более чем на 1. Однако, наша цель — получить асимптотическую сложность поиска в $O(\log N)$, и, как станет видно далее, для этой цели даже сбалансированность — избыточное требование.

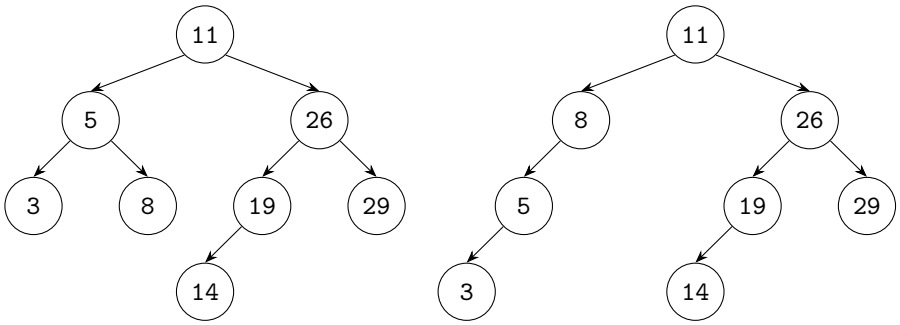

Рис. 1.4: Пример сбалансированного (слева) и несбалансированного (справа) двоичных деревьев поиска, хранящих одинаковую коллекцию.

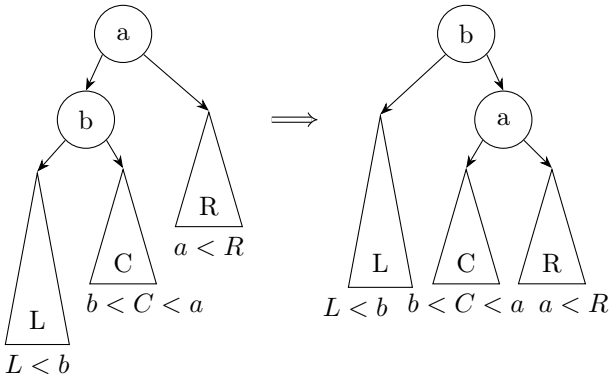

Рис. 1.5: Пример операции поворота «малое правое вращение». Треугольники обозначают поддеревья.

рис. 1.4 приведён пример сбалансированного и несбалансированного двоичных деревьев поиска. К сожалению, приведённые выше примеры операций вставки и удаления очевидным образом портят балансировку дерева. Тогда нужно либо отказаться от модификации дерева, изначально построенного сбалансированным, либо изменить эти операции таким образом, чтобы гарантировать сбалансированность дерева после применения каждой операции.

Одним из примеров последнего подхода являются *АВЛ-деревья* (Адельсон-Вельский и Ландис 1962), названные так в честь авторов — советских математиков Г.М. Адельсон-Вельского и Е.М. Ландиса, которые предложили способы вставки и удаления элемента, не портящие балансировку дерева. Предложенный ими метод базируется на следующих основных идеях. Во-первых, предлагается в каждый узел дополнительно записать *показатель сбалансированности* — разницу между высотами левого и правого поддеревьев. Согласно определению, этот показатель может принимать всего три

различных значения $\{-1, 0, +1\}$.

Во-вторых, после вставки или удаления элемента выполняется обратный проход от листа к корню по ранее запомненному маршруту. В момент этого прохода для каждого узла удаётся последовательно определить, как изменился его показатель сбалансированности (доказано, что эту операцию можно выполнить за константное время), и, если показатель баланса стал -2 или $+2$, то выполнить одну из четырёх описанных авторами операций балансировки, называемых вращениями и требующих константного времени. Операции балансировки состоят в переподчинении поддеревьев другим узлам, пример одной из операций приведён на рис. 1.5. Таким образом, операции вставки и удаления в АВЛ-дереве, включая последующую перебалансировку дерева, требуют $O(h)$ операций.

Оказывается, что можно ослабить требования балансировки, сохранив асимптотическую сложность в $O(\log N)$ операций для удаления, вставки и поиска. Примером являются *красно-чёрные деревья*, название которым дали американские исследователи Л. Гибас и Р. Седжвик (Guibas и Sedgewick 1978). Для красно-чёрных деревьев гарантируется, что минимальная высота листа и максимальная высота листа различаются не более чем в два раза, и удаётся показать, что асимптотическая сложность основных операций $O(\log N)$. Можно сказать, что красно-чёрные деревья аналогичны АВЛ-деревьям. Во-первых, предлагается хранить в каждом узле *цвет* (один бит информации — «красное» или «чёрное»). Цвета задаются не произвольно, красно-чёрное дерево подчиняется набору инвариантных требований, например, если узел красный, то оба потомка чёрные, и т.п.

Во-вторых, при обратном проходе удаётся сформулировать набор правил перекрашивания (сравните с подсчётом показателя сбалансированности) и правил перебалансировки (их в этот раз оказывается девять). Суть в том, что, как и в случае с АВЛ-деревьями, на каждый узел по пути обратного прохода потребуется константное время. Перечислять здесь полный набор условий и правил было бы утомительно для читателя, как и в случае с АВЛ-деревом.

Интересно, что для *случайного двоичного дерева поиска* средняя глубина (в отличие от максимальной глубины, которая рассматривалась в двух предыдущих случаях) тоже оказывается пропорциональна $\log N$. Случайным деревом называется дерево, в котором корень выбирается равновероятно, затем коллекция элементов разделяется на элементы, оказавшиеся меньше корня (левое поддерево) и больше корня (правое поддерево). Для левого и правого поддерева операция повторяется.

Деревья поиска наиболее часто используются в качестве *ассоциативного массива*. В таких случаях данные x представляются в виде пары (k, v), состоящей из *ключа* k и *значения* v. Ключ и значение неравнозначны. Для ключей k должна быть определена операция сравнения, и они используются для построения дерева, а значения v просто дополнительно приписываются узлам. Результатом является возможность эффективно отыскать v', зная ключ k', но не наоборот — зная v', отыскать соответствующий ему ключ k' можно только полным перебором дерева. Примером может служить те-

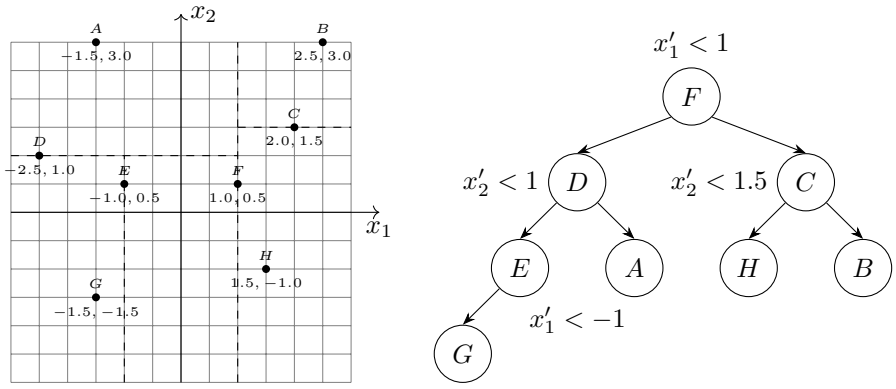

Рис. 1.6: Схематичное изображение k-мерного дерева для $\mathbf{R}^2$ (слева) и получаемого разбиения пространства (справа).

лефонная книга. Когда на телефон поступает звонок с номера k', мы заинтересованы в том, чтобы быстро найти в телефонной книге имя этого абонента v' (или обнаружить, что номер неизвестный) и показать пользователю сообщение о поступающем вызове.

Кроме того, из-за упорядоченной структуры деревья позволяют эффективно находить все элементы, удовлетворяющие условиям $a \leq x$, $x \leq b$, $x \in [a, b]$, $x \in [x_0 - \epsilon, x_0 + \epsilon]$, а так же выполнять поиск ближайшего снизу или ближайшего сверху значений к заданным элементам. Но для эффективного выполнения таких операций дерево должно быть дополнительно *прошито*, т.е. в каждый узел с пустым правым поддеревом записывается адрес узла, содержащего данные следующие за этим узлом, а в каждый узел с пустым левым поддеревом записывается адрес узла, содержащего данные предшествующие этому узлу.

Если вспомнить пример про астрономические каталоги из начала этого раздела, то возможность эффективного поиска ближайших соседей в заданной окрестности выглядит особенно привлекательно. Жаль только, что координаты представляют собой двумерный вектор, и непонятно, как их упорядочивать, чтобы представить в виде дерева поиска.

k-мерные деревья

Элементы многомерного пространства $\mathbf{R}^k$, $k > 1$ представляют особый интерес, но для хранения таких данных используются специализированные виды деревьев: *k-мерные деревья*[16], деревья квадрантов, R-деревья и т.д.

Рассмотрим в качестве примера принцип организации многомерных данных в k-мерных деревьях (Bentley 1975). Этот подход пригодится нам в будущем в разделах про машинное обучение. Во-первых, k-мерное дерево — двоичное. Во-вторых, для заданной размерности пространства k все

[16] k-d trees

существующие уровни двоичного дерева пронумерованы циклически от 1 до k. На практике номер уровня не нужно дополнительно хранить, так как он всегда может быть найден как побочный результат во время обхода дерева. В-третьих, на каждом уровне сравнивается только компонента векторов, соответствующая циклическому номеру уровня. Например, в случае плоскости в левом поддереве окажутся точки, которые находятся левее корневой точки (неважно, выше или ниже), а в правом поддереве — правее. На следующем уровне два поддерева будут сформированы из точек, которые лежат выше или ниже точки корня. Пример k-мерного дерева приведён на рис. 1.6. В многомерном пространстве k-мерные деревья позволяют эффективно выполнять операцию поиска элемента, и, кроме того, операцию поиска ближайших соседей ($O(\log N)$ в среднем).

Стоит сказать несколько слов о балансировке k-мерных деревьев. Из заданного набора точек достаточно просто построить сбалансированное дерево: для этого на каждом этапе достаточно выбрать медиану среди значений той координаты векторов, которая рассматривается на данном уровне. Однако, в отличие от одномерного случая, операции удаления и записи представляют собой нетривиальную проблему, поэтому удобно, когда модифицировать коллекцию не нужно. Но стоит отметить, что существуют и самобалансирующиеся варианты k-мерных деревьев, и стратегии с отложенной балансировкой, когда дерево перестраивается целиком или большими частями один раз в несколько вставок или удалений.

1.4.4 Хеш-таблицы

Зададимся вопросом, можно ли осуществить поиск по значению (или по ключу), вставку и удаление элемента за константное время $O(1)$. Вспомним, что чтение и запись элемента массива по его индексу i — это единственная операция с константной сложностью среди описанных выше.

Рассмотрим вначале упрощённый, но малопрактичный способ, который называется *таблица прямой адресации*. Предположим, что мы хотим хранить подмножество $X \subset \mathbf{Z}_{\geq \mathbf{0}}$ неотрицательных целых чисел. Тогда мы могли бы организовать массив, значениями которого являются только 0 и 1, по следующему принципу: по индексу i стоит 1, если $i \in X$, или 0 в противном случае. При таком подходе для поиска элемента по значению x достаточно было бы обратиться к памяти по адресу $a_0 + i \cdot 1$, что занимает константное время. Аналогичным образом мог бы быть реализован ассоциативный массив. Для иллюстрации вспомним пример про телефонную книгу из раздела 1.4.3: в элементе номер i таблицы прямой адресации будем хранить либо адрес строки с именем абонента с номером телефона i, либо признак отсутствия номера в телефонной книге. Очевидно, что поиск, добавление и удаление записи с известным номером телефона займёт константное время.

Этот способ имеет два серьёзных недостатка, значительно сужающих область его практического применения. Во-первых, не очевидно, как этот подход адаптировать для хранения, например, строк. Что, если бы мы захотели построить обратную телефонную книгу и уметь эффективно находить

номер, зная имя абонента? Во-вторых, способ может потребовать огромного количества памяти, порядка

$$\max_{i \in X}\{i\} - \min_{i \in X}\{i\},$$

большая часть которой, оказалась бы заполнена признаками отсутствия данных. Например, если телефонный номер записан в виде десятизначного десятичного числа, то потребуется 10^{10} ячеек таблицы прямой адресации. Пусть одна ячейка занимает 8 байт, тогда таблица потребовала бы около 75 GB памяти. Понятно, что в телефонной книге у одного человека может быть от силы тысяча контактов, т.е. эффективность использования памяти составляет порядка 10^{-7}.

Оказывается, что эти недостатки можно преодолеть следующим образом: назовём *хеш-функцией*[17] такую детерминированную функцию $i = h(x)$, которая любому x любой требуемой природы (целому числу, вектору, действительному числу или даже строке) сопоставляет целочисленный индекс i в разумном интервале значений (например, $[0, 2^{32})$ или $[0, 2^{64})$). При этом для любого распределения x соответствующее распределение $i = h(x)$ должно быть как можно ближе к равномерному. Желательно также, чтобы хеш-функция вычислялась сравнительно просто и быстро.

Например, возможный вариант хеш-функции для натурального числа x:

$$h(k) = x \mod p, \tag{1.13}$$

где p некоторое простое число, а операция $x \mod p$ означает взятие остатка от деления x на p. Понятно, что хеш-функция не может быть определена единственным образом, поэтому возможен, например, такой альтернативный вариант:

$$h(k) = \lfloor m\{x \cdot A\} \rfloor, \tag{1.14}$$

где $A \in (0, 1)$ и $m \in \mathbf{Z}_+$ — некоторые параметры, $\{\cdot\}$ обозначает дробную часть числа, а $\lfloor \cdot \rfloor$ обозначает округление вниз до целого.

Пример хеш-функции для строки:

$$h(\mathbf{x}) = \left(\sum_{j=0}^{L-1} p^j x_j \right) \mod q, \tag{1.15}$$

где p и q некоторые простые числа, а x_j — числовой код символа номер j в строке из L символов. Составление хорошей хеш-функции — процесс творческий, но, к счастью, на любой случай уже было придумано и реализовано достаточное количество эффективных вариантов.

Теперь следующее улучшение таблицы прямой адресации будем называть *хеш-таблицей*[18]. Во-первых, зафиксируем некоторую хеш-функцию

[17] *hash function*
[18] *hash table*

$h(x)$ для интересующего нас типа элементов. Во-вторых, зафиксируем заранее таблицу из M одинаковых по размеру ячеек[19], в которых хранится либо некоторый элемент x, либо признак его отсутствия. В-третьих, договоримся, что будем хранить элемент со значением x только в ячейке с номером $i' = h(x)$. На этот раз x может иметь любой тип данных, так как преобразуется в индекс i' с помощью хеш-функции. Если хеш-функция $h(x)$ принимает значения в интервале $[0, 2^\omega)$, то удобно выбирать размер таблицы как степень двойки $M = 2^m$, где $m \leq n$. Можно доказать, что если случайное число распределено равномерно в интервале $[0, 2^\omega)$, то его остаток от деления на $2^{\omega-1}$ будет равномерно распределён в интервале $[0, 2^{\omega-1})$. Таким образом, из одной хеш-функции порождается другая хеш-функция, адаптированная под требуемый размер таблицы. Вернёмся к примеру с телефонной книгой: если мы используем вместо таблицы прямой адресации хеш-таблицу размером $M = 1024$, то эффективность использования памяти будет близка к 1.

На первый взгляд, сложность операций поиска, удаления и вставки элемента в хеш-таблицу по-прежнему составляет $O(1)$, как это было в случае с таблицей прямой адресации. Однако, чаще всего хеш-функция — это сюръективное отображение, т.е. найдутся x_1 и x_2 такие, что $h(x_1) = h(x_2)$. А значит, мы сможем записать в хеш-таблицу либо x_1, либо x_2, но никак не x_1 и x_2 одновременно. Эта ситуация называется *коллизией*.

Одним из способов разрешения коллизий является *разрешение коллизий методом цепочек*, т.е. с помощью списков. Допустим, что каждый элемент нашей таблицы из M элементов является односвязным списком (возможно пустым), и этот список уже хранит реальные значения x (либо пары ключ-значение k, v). При этом список номер i хранит только элементы с одинаковой хеш-функцией. Теперь все интересующие нас операции выполняются в два этапа. Чтобы найти элемент x (либо найти значение v по ключу k), следует сначала за $O(1)$ операций обратиться к ячейке $i' = h(x)$ (либо $i' = h(k)$), а затем пройти весь соответствующий список из $N_{i'}$ элементов и сопоставить каждый его элемент с искомым, что займёт $O(N_{i'})$ операций. Аналогично, чтобы удалить или добавить элемент x, нужно сначала обратиться к списку в ячейке $i' = h(x)$, затем найти в нём элемент x и далее удалить или добавить новый элемент в этот список. В худшем случае $N_{i'} = N$ и, следовательно, сложность рассматриваемых операций составляет $O(N)$, однако, хеш-таблица всё ещё обеспечивает константную сложность операций *в среднем*.

Известно приблизительное выражение для вероятности того, что из N чисел, равномерно распределённых от 0 до M не включительно, хотя бы два совпадают:

$$p = 1 - \exp\left(-\frac{N(N-1)}{2M}\right). \tag{1.16}$$

Допустим, что мы выбрали количество ячеек таким образом, что $M \gg N$. В этом случае видно, что вероятность коллизий окажется пренебрежимо

[19] Часто ячейку называют *bucket* — корзинка.

мала. Понятно, что в этом случае хеш-таблица будет обеспечивать быстрый доступ, но с низкой эффективностью использования памяти, так как доля использованных ячеек составит $\frac{N}{M} \to 0$.

В случае, если мы выбрали количество ячеек так, что $M \ll N$, то вероятность коллизий будет близка к 1, и каждая ячейка будет хранить очень длинный список из $\frac{N}{M}$ элементов. В таком случае эффективность использования памяти будет высокой, а скорость доступа, наоборот, линейной по числу элементов.

На практике в результате вставок и удалений элементов количество хранимых записей N постоянно меняется, поэтому часто M подбирается динамически, чтобы обеспечить оптимальный баланс между скоростью работы и использованием памяти: в моменты, когда заполненность хеш-таблицы сильно меняется в ту или иную сторону, происходит перебалансировка таблицы. Создаётся новая таблица размером $2\,M$ (или $\frac{1}{2}M$) с новой хеш-функцией $h'(x)$, в которую перекладываются все элементы.

Стоит отметить интересный способ борьбы с коллизиями в случае, когда область допустимых значений x (или ключей k) невелика. Например, нам попались названия химических элементов, закодированные в виде строк. В таком случае, во-первых, заранее можно перечислить все возможные варианты, и, во-вторых, их число будет с запасом меньше, чем $2^8 = 256$. Оказывается, существуют алгоритмы, позволяющие построить *идеальную хеш-функцию*[20], которая будет *взаимооднозначно* отображать допустимое множество ключей в множество индексов.

[20] perfect hash function

Глава 2

Метод наименьших квадратов

Для современного физика вполне естественно, что физическая реальность воспринимается и интерпретируется на математическом языке. С одной стороны, измерения различных физических величин, проводимые в экспериментах, по сути своей числа. С другой стороны, физические величины входят в уравнения, выражающие фундаментальные физические законы.

Математической моделью, или просто *моделью*, как раз и называется набор математических утверждений, описывающих конкретный эксперимент или явление. Слово модель используется, чтобы подчеркнуть эквивалентность математических абстракций и материального физического мира, при рассмотрении через призму физических теорий.

Чаще всего модель используют двумя способами. Во-первых, можно поставить *прямую задачу* — предсказать значение измеряемых физических величин до (или вместо) проведения эксперимента, зафиксировав параметры модели. Например, зная длину подвеса математического маятника L и ускорение свободного падения g, можно предсказать период малых колебаний маятника T.

Во-вторых, можно поставить *обратную задачу*, которая состоит в том, что проведя измерения некоторых физических величин, нужно оценить какими должны были бы быть параметры модели. Например, можно попытаться оценить ускорение свободного падения g, измерив периоды малых колебаний математических маятников T_i с разной длиной подвеса L_i. Если бы мы научились использовать маятник как прибор для точного измерения ускорения свободного падения, то можно было бы сформулировать более сложный, но и более реалистичный пример обратной задачи: измерив ускорение свободного падения в нескольких точках на поверхности Земли g_{ij} нужно восстановить объёмное распределение плотности вещества под поверхностью $\rho(\mathbf{x})$. В таком случае области с более низкой или более высокой плотностью указывали бы нам на места возможного расположения

полезных ископаемых. В рамках данного курса мы сталкиваемся преимущественно с обратными задачами.

В физике модель можно построить двумя основными способами. Во-первых, применив фундаментальные физические законы к конкретным условиям, например решив те или иные системы уравнений с нужными начальными или граничными условиями. В этом случае параметрами модели являются конкретные физические величины, непосредственно не измеряемые в рамках эксперимента, но определяемые в результате решения обратной задачи. Значения этих физических величин имеют самостоятельную ценность (очевидно, что масса шарика или сопротивление резистора являются присущими им характеристиками и не могут произвольно изменяться между экспериментами) и могут быть затем использованы в других моделях как известные параметры. Кроме того, фундаментальные физические законы допускают экстраполяцию предсказаний модели в разумных пределах. Это означает, что определив параметры модели при одних условиях проведения эксперимента, мы можем предсказывать интересующие нас значения в рамках применимости модели, которые могут быть шире рамок проведённого эксперимента. При сильном рассогласовании измерений и предсказаний модели фундаментальные физические законы как правило не подвергаются сомнению[1], однако конкретная модель может быть подвергнута ревизии на предмет неучтённых ранее эффектов. Так, например, может оказаться, что колебания температуры в помещении лаборатории, которые изначально считались пренебрежимо малыми, существенным образом сказываются на результате измерений, и температурные зависимости должны быть включены в модель, а термометр должен быть включён в экспериментальную установку.

Во-вторых, модель можно построить эмпирическим способом, т.е. исходя из данных измерений, имеющихся в наличии. Такой способ часто используется для так называемых калибровок, когда точное применение законов физики упирается либо в сложность их применения, либо в недостаток данных. Например, у нас есть термопарный датчик температуры: мы измеряем напряжение на его контактах, которое, как мы знаем, зависит от температуры. Мы теоретически могли бы построить полную модель датчика и установить связь между температурой и напряжением, но для этого нам потребовался бы слишком детальный анализ и знание всех дефектов изготовления конкретного датчика, которые пришлось бы исследовать отдельно. Гораздо проще окунуть конкретный датчик в ёмкости с веществами при заданной температуре и исследовать его отклик. Такими веществами могут быть кипящий жидкий азот, ледяная вода, и т.п. Построив несколько точек на графике, мы бы заметили, что зависимость практически линейная, и можно определить коэффициенты пересчёта из единиц измерения напряжения в единицы измерения температуры.

Или, например, нас интересует измерение положений спектральных ли-

[1] Кроме небольшого числа известных случаев пост-фактум описанных в литературе по философии и методологии науки.

ний некоторого вещества с помощью спектрографа. В спектрографе, однако, мы получаем цифровое изображение спектра, где пиксели пронумерованы дискретными числами, а вовсе не физической длинной волны. Вооружившись законами оптики и сопротивления материалов, мы могли бы тщательно измерить все расстояния внутри нашего прибора, положение самого детектора изображений, всех оптических элементов, и т.п. Вероятно, потребуется учесть, как эти расстояния будут меняться в зависимости от температуры в лаборатории. Но в конечном счёте, выполнив сложные расчёты, мы теоретически нашли бы закон, который бы каждому номеру пикселя сопоставлял длину волны. На практике, обычно применяются методы калибровки: получим спектр известного вещества, посмотрим на какие пиксели придутся линии с известными длинами волн, и получим соотношение, которое бы для каждого номера пикселя предсказывало длину волны в физических единицах.

Полученные нами параметры эмпирической модели не будут иметь самостоятельного физического смысла, и пригодятся нам только для предсказаний конкретной модели, причём в условиях не выходящих за рамки, при которых проводилась калибровка. В таком случае говорят, что модель способна на интерполяцию. В отличии от первого случая, требуется убедиться, что предложенная эмпирическая модель, которая фактически выбирается произвольно, хорошо описывает данные. Обычно используются те или иные критерии проверки статистических гипотез или метрики качества предсказания модели. Кроме того, не всегда понятно, как подобрать модель, которая хорошо описывает данные, в случаях, когда зависимость не очевидна на глаз, однако методы машинного обучения могут быть существенным подспорьем в таких ситуациях.

2.1 Линейные модели

Вообразим, что некоторый физический процесс или эксперимент можно описать простой многомерной линейной моделью следующего вида

$$A\theta = \mathbf{b} \tag{2.1}$$

где $\theta \in \mathbf{R}^m$, $\mathbf{b} \in \mathbf{R}^n$, обычно оказывается, что $n > m$. Здесь вектор θ — некоторые параметры модели, матрица A — известные обстоятельства эксперимента, вектор $\mathbf{b}$ — измеряемые (наблюдаемые) физические величины. Матрица A считается известной, а если задать вектор θ, то можно рассчитать вектор $\mathbf{b}$, т.е. решить прямую задачу или, как иногда говорят, построить модельное предсказание. Заметим, что указанная линейная модель могла бы получиться из обоих источников, упомянутых выше.

Например, принимаемый некоторым устройством на каком-то телескопе поток фотонов от некоторой звезды F_i может быть выражен следующим уравнением:

$$\ln F_i = -kM_i + C, \quad 1 \leq i \leq n, \tag{2.2}$$

где k — способность атмосферы ослаблять свет, M_i — *воздушная масса* (длина отрезка на луче зрения от наблюдателя до верхней границы атмосферы), C — некоторая инструментальная константа, общая для всех измерений выполненных на этом приборе и включающая в себя его геометрию, и эффективность способности регистрировать свет. В этом случае вектор параметров:

$$\theta \equiv \begin{bmatrix} -k \\ C \end{bmatrix}. \tag{2.3}$$

Матрица A принимает следующий вид:

$$A \equiv \begin{bmatrix} M_1 & 1 \\ M_2 & 1 \\ \vdots & \vdots \\ M_n & 1 \end{bmatrix}. \tag{2.4}$$

А вектор предсказания модели:

$$\mathbf{b} \equiv \begin{bmatrix} \ln F_1 \\ \ln F_2 \\ \vdots \\ \ln F_n \end{bmatrix}. \tag{2.5}$$

Обратим внимание, что вектор параметров θ может быть функцией от физических величин, так, например, в нашем случае первый компонент взят со знаком минус. Понятно, что если при решении обратной задачи нам удастся отыскать вектор θ, то и физические величины можно будет вычислить. Пусть, например, мы хотим изучить внутренние характеристики приёмника изображений на базе прибора с зарядовой связью (ПЗС), тогда нам пришлось бы анализировать статистические характеристики сигнала в отдельном пикселе (дисперсию $\sigma^2_{\Delta F_i}$) при разных интенсивностях засветки F_i. Если излучение подчиняется пуассоновскому закону, тогда можно получить следующую линейную модель:

$$\sigma^2_{\Delta F_i} = 2\frac{\sigma_r^2}{g^2} + \frac{2}{g}F_i, \quad 1 \le i \le n, \tag{2.6}$$

где g — *коэффициент усиления*, σ_r^2 — *шум считывания*, две интересующие нас характеристики прибора, которые имеют для нас самостоятельную ценность, так как мы намерены использовать их впоследствии для априорной оценки точности получаемых измерений. Видно, что в этом случае вектор параметров:

$$\theta \equiv \begin{bmatrix} 2\sigma_r^2 g^{-2} \\ 2g^{-1} \end{bmatrix}. \tag{2.7}$$

2.2 Метод наименьших квадратов

Понятно, что необходимо как-то уметь решать обратную задачу для линейных моделей. Одна из проблем при решении обратных задач заключается в

том, что любые измерения даны нам с некоторой погрешностью, величина которой остаётся для нас неизвестной. Таким образом вместо вектора $\mathbf{b}$ мы измеряем некоторый вектор $\hat{\mathbf{b}} = \mathbf{b} + \epsilon$, где вектор ϵ обозначает ту самую неизвестную погрешность. Одного этого достаточно, чтобы исходная система линейных уравнений оказалась неразрешимой, ведь в правую часть мы теперь подставляем реализацию некоторой случайной величины.

Метод (принцип) наименьших квадратов заключается в том, что исходная обратная задача заменяется на следующую задачу:

$$\hat{\theta} = \arg\min_{\theta} \|A\theta - \hat{\mathbf{b}}\|^2, \tag{2.8}$$

в такой формулировке мы можем вместо $\mathbf{b}$ использовать наши измерения с погрешностью $\mathbf{b} + \epsilon$. Такой способ был независимо предложен как минимум двумя знаменитыми математиками XIX века: Лежандром и Гауссом (см. Stigler 1981).

В самом общем случае метод наименьших квадратов следует из здравого смысла и представлений о том, как устроены измерения. В частных случаях, метод наименьших квадратов может быть выведен из других принципов, например принципа максимального правдоподобия, о чём пойдёт речь в разделе 5.

Так или иначе, сейчас замена исходной задачи на задачу наименьших квадратов производится волюнтаристическим образом. Конечно, при таком положении дел сразу же возникает вопрос как оценка $\hat{\theta}$ соотносится с истинными параметрами модели? В некоторых частных случаях, введя дополнительные предположения о характере ошибки измерений, удаётся продемонстрировать ряд свойств оценки параметров, получаемой методом наименьших квадратов, таким образом оправдав применение метода. Рассмотрим их по порядку.

Для начала найдём решение задачи линейных наименьших квадратов в виде формулы.

$$\begin{aligned} \|A\theta - \hat{\mathbf{b}}\|^2 &= (A\theta - \hat{\mathbf{b}}, A\theta - \hat{\mathbf{b}}) = \\ &= (A\theta, A\theta) - 2(A\theta, \hat{\mathbf{b}}) + (\hat{\mathbf{b}}, \hat{\mathbf{b}}) \end{aligned} \tag{2.9}$$

вычисляя градиент выражения по θ и приравнивая его к нулю (можно доказать, что если $\det A^T A \neq 0$, то функция, которую мы минимизируем выпуклая, а значит обладает единственным минимумом), получаем следующие уравнения

$$A^T A\hat{\theta} = A^T \hat{\mathbf{b}} \tag{2.10}$$

Эти уравнения называют *нормальными уравнениями*. В отличии от исходной линейной модели, матрица $A^T A$ квадратная. Вопрос равенства нулю определителя $\det A^T A$ будет рассмотрен в разделе 2.4, но если предположить, что он отличен от нуля, то можно записать выражения для оценки параметров модели

$$\hat{\theta} = (A^T A)^{-1} A^T \hat{\mathbf{b}} \tag{2.11}$$

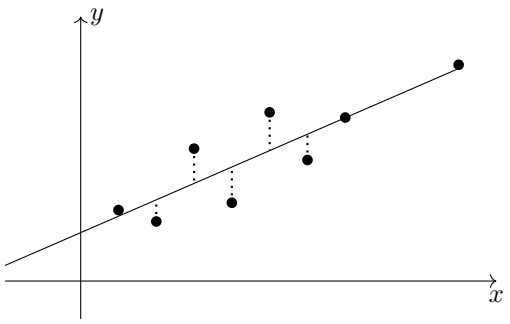

Рис. 2.1: Применение метода наименьших квадратов для определения параметров модели $\theta_1 x_i + \theta_2 = \hat{y}_i$. Точками показаны положения выборки $(x_i, \hat{y}_i)$; сплошной линией показана модельная кривая $y = \hat{\theta}_1 x + \hat{\theta}_2$ на основе оценки параметров, полученной методом наименьших квадратов; пунктирные линии показывают минимизируемое расстояние между измерениями и предсказаниями модели.

Данное выражение[2] не единственный способ вычисления $\hat{\theta}$, и, в ряде случаев, не самый практичный, но он незаменим для теоретического анализа свойств решения задачи. На Рис. 2.1 графически показан пример применения метода наименьших квадратов для определения параметров линейной модели.

Итак, предположим, что ошибка ϵ в среднем равна нулю: $\mathrm{E}[\epsilon] = 0$, тогда

$$\begin{aligned} \mathrm{E}\left[\hat{\theta}\right] &= (A^T A)^{-1} A^T \mathrm{E}\left[\hat{\mathbf{b}}\right] = \\ &= (A^T A)^{-1} A^T \left(\mathbf{b} + \mathrm{E}[\epsilon]\right) = \\ &= (A^T A)^{-1} A^T \mathbf{b} = \\ &= (A^T A)^{-1} A^T A\theta = \theta. \end{aligned} \tag{2.12}$$

Откуда видно, что в предположении невырожденности матрицы $A^T A$ ($\det A^T A \neq 0$), оценка $\hat{\theta}$ является несмещённой. Иными словами, если эксперимент будет повторяться вновь и вновь, то среднее получаемых оценок будет стремиться к истинным значением параметров.

Предположим далее, что ошибка измерений независима и обладает одинаковой дисперсией, т.е. $\operatorname{cov} \epsilon = \sigma^2 I$, тогда

$$\hat{\theta} - \mathrm{E}\left[\hat{\theta}\right] = (A^T A)^{-1} A^T \epsilon, \tag{2.13}$$

$$\begin{aligned} \operatorname{cov} \hat{\theta} &= (A^T A)^{-1} A^T \operatorname{cov} \epsilon ((A^T A)^{-1} A^T)^T = \\ &= \sigma^2 (A^T A)^{-1} \end{aligned} \tag{2.14}$$

[2]Матрицу $(A^T A)^{-1} A^T$ иногда называют *псевдообратной матрицей* для матрицы A.

Мы нашли матрицу ковариации оценки параметров методом наименьших квадратов. Однако, тут есть две проблемы: во-первых дисперсия σ^2 обычно заранее не известна; во-вторых для интерпретации ошибки параметров в терминах доверительных интервалов нужно дополнительно потребовать, что ϵ распределён тем или иным образом (чаще всего предполагается нормальное распределение).

Попытаемся оценить дисперсию измерений σ^2, для этого введём оценку ошибки измерений $\hat{\epsilon}$, которую часто называют невязкой:

$$\hat{\epsilon} = \hat{\mathbf{b}} - A\hat{\theta} = (I - A(A^TA)^{-1}A^T)\hat{\mathbf{b}}. \tag{2.15}$$

Среднее такого вектора равно нулю, а матрица ковариации в предположении $\operatorname{cov}\epsilon = \sigma^2 I$:

$$\operatorname{cov}\hat{\epsilon} = \sigma^2(I - A(A^TA)^{-1}A^T). \tag{2.16}$$

Обратим внимание, на средний квадрат нормы невязки $\|\hat{\epsilon}\|^2$:

$$\mathrm{E}\left[\|\hat{\epsilon}\|^2\right] = \operatorname{tr}\operatorname{cov}\hat{\epsilon} = \sigma^2 \operatorname{tr}\left(I - A(A^TA)^{-1}A^T\right) = \sigma^2(n-m) \tag{2.17}$$

Откуда получается окончательная оценка матрицы ковариации оценки параметров методом наименьших квадратов:

$$\operatorname{cov}\hat{\theta} = \hat{\sigma}^2(A^TA)^{-1} = \frac{\|\hat{\epsilon}\|^2}{n-m}(A^TA)^{-1}, \tag{2.18}$$

где $n-m$ часто называют *числом степеней свободы*.

Если предположить, что вектор ϵ распределён в соответствии с многомерным нормальным распределением с нулевым средним и матрицей ковариации $\sigma^2 I$, тогда из свойств нормального распределения следует, что вектор $\hat{\theta}$ тоже распределён нормально, причём его среднее и дисперсию мы уже вычислили выше. Теперь мы не просто можем охарактеризовать дисперсию оценки интересующих нас параметров модели, но и легко вычислить вероятность, с которой доверительный интервал накрывает истинное значение параметра.

2.3 Метод взвешенных наименьших квадратов

На практике разные компоненты вектора $\mathbf{b}$ могут иметь различную физическую размерность, а значит предположение о $\operatorname{cov}\epsilon = \sigma^2 I$ заведомо не выполняется. В случае неодинаковых ошибок оказывается полезным следующее обобщение метода наименьших квадратов:

$$\hat{\theta} = \arg\min_{\theta}\left(A\theta - \hat{\mathbf{b}}, W^{-1}\left(A\theta - \hat{\mathbf{b}}\right)\right), \tag{2.19}$$

где матрица $W = W^T$ называют *матрицей весов*. Фактически, мы заменили норму линейного пространства на некоторую другую. Интуитивно понятно, что если данные состоят из набора более точных измерений и менее точных

измерений, то параметры модели нужно подбирать так, чтобы предсказания точных измерений имели бы меньшую невязку, чем предсказания для менее точных.

Найдём значение матрицы весов W следующим образом: во-первых, новая оценка параметров $\hat{\theta}$ должна быть несмещённой, во-вторых, по возможности, иметь наименьшую дисперсию.

Для начала, по аналогии с предыдущими вычислениями мы можем установить, что

$$\hat{\theta} = (A^T W^{-1} A)^{-1} A^T W^{-1} \hat{\mathbf{b}}, \tag{2.20}$$

следовательно $\mathrm{E}\left[\hat{\theta}\right] = \theta$, т.е. оценка по прежнему не смещённая при любом выборе W.

Затем, аналогичным образом найдём матрицу ковариации оценки $\hat{\theta}$:

$$\operatorname{cov}\hat{\theta} = \left((A^T W^{-1} A)^{-1} A^T W^{-1}\right) (\operatorname{cov}\epsilon) \left((A^T W^{-1} A)^{-1} A^T W^{-1}\right)^T, \tag{2.21}$$

если принять, что $W = \operatorname{cov}\epsilon$, то выражение значительно упрощается:

$$\operatorname{cov}\hat{\theta} = (A^T W^{-1} A)^{-1}. \tag{2.22}$$

Докажем от противного, что этот выбор гарантирует наименьшую ошибку оценки параметров. Предположим, что существует некоторая альтернативная линейная оценка параметров:

$$\hat{\theta}' = (A^T W^{-1} A)^{-1} A^T W^{-1} \hat{\mathbf{b}} + D\hat{\mathbf{b}}, \tag{2.23}$$

где D некоторая новая дополнительная матрица, из требования несмещённости оценки следует $DA = 0$. Тогда

$$\operatorname{cov}\hat{\theta}' = (A^T W^{-1} A)^{-1} + DWD^T, \tag{2.24}$$

где второе слагаемое — положительно определённая матрица, т.е. для любого $D \neq 0$ дополнительная добавка к ошибке оценки параметров. Итак, оценка параметров методом взвешенных наименьших квадратов является оценкой с наименьшей дисперсией в случае $W = \operatorname{cov}\epsilon$. Кстати, оценка параметров обычным методом наименьших квадратов тоже обладает наименьшей дисперсией среди всех линейных оценок, чтобы показать это, достаточно положить $W = I$.

Другим важным частным случаем является случай когда W диагональная матрица, это случай независимых, но неравноточных измерений. В таком случае вычисление обратной матрицы W^{-1} выполняется особенно удобно.

В самом общем случае, если вспомнить, что $W = W^T$, то можно представить $W^{-1} = NN^T$, где N некоторая матрица. Одним из вариантов (но не единственным) вычисления матрицы N является алгоритм разложения Холецкого. В таком случае технически удобно свести задачу вычисления оценки параметров к задаче невзвешенных наименьших квадратов:

$$\hat{\theta} = \arg\min_{\theta} \|N^T A\theta - N^T \hat{\mathbf{b}}\|^2, \tag{2.25}$$

перед последующими вычислениями вектор правой части $\hat{\mathbf{b}}$ и матрица A домножаются слева на N^T.

К сожалению, матрицу ковариации измерений $\operatorname{cov}\epsilon = W$ следует знать заранее. Если нет никаких теоретических предпосылок, позволяющих вычислить W, то одним из самых простых способов является вычисление выборочной матрицы ковариации измерений $\hat{\mathbf{b}}$, где выборка формируется из повторений всех измерения несколько раз. Однако такой способ сопряжён и с практическими трудностями, ведь для уверенного определения выборочной матрицы ковариации следует получить много реализаций вектора $\hat{\mathbf{b}}$, по крайней мере много больше n^2. Недостаточная точность оценки W приведёт к численным трудностям при последующем обращении матрицы или при её факторизации.

Полезно рассмотреть случай, когда $\operatorname{cov}\epsilon = \sigma^2 W_0$, где W_0 известная матрица, а σ^2 неизвестный масштабирующий фактор. В таком случае удобно положить матрицу весов $W = W_0$, а затем оценить σ^2 по аналогии со случаем метода наименьших квадратов без весов. Оценка ошибки оценки параметров линейной модели будет задаваться следующей формулой:

$$\operatorname{cov}\hat{\theta} = \frac{\|\hat{\epsilon}\|^2}{n-m}(A^T W_0^{-1} A)^{-1}, \tag{2.26}$$

где $\hat{\epsilon}$ теперь вычисляется как

$$\hat{\epsilon} = \hat{\mathbf{b}} - A\hat{\theta} = (I - A(A^T W_0^{-1} A)^{-1} A^T W_0^{-1})\hat{\mathbf{b}}. \tag{2.27}$$

2.4 Плохо обусловленные и некорректные задачи

При получении оценки (2.11) мы предполагали, что $\det A^T A \neq 0$, а случай когда матрица $A^T A$ вырождена остался пока не рассмотрен. Стоит сразу сказать, что это не исключительный случай, а скорее предельный. Действительно, технически проверить условие $\det A^T A = 0$ оказывается затруднительным из-за наличия ошибок округления при численных расчётах. Так, вычисление определителя квадратной матрицы потребует порядка $O(n^3)$ операций умножения, которые внесут ошибки округления и в ответе почти наверняка получится некоторое малое число. Поэтому, как часто и бывает в численных методах, задачи с $\det A^T A = 0$ для нас на практике не отличимы от некоторого более широкого класса задач, где матрица $A^T A$ формально невырожденная.

Напомним, что *числом обусловленности* квадратной матрицы H называется

$$k \equiv \frac{\lambda_{\max}}{\lambda_{\min}}, \tag{2.28}$$

где $\lambda_{\max}$, $\lambda_{\min}$ — максимальное и минимальное собственные значения мат-

рицы H. Можно показать, что

$$\frac{\|\delta\hat{\theta}\|}{\|\hat{\theta}\|} < k\frac{\|\delta\hat{\mathbf{b}}\|}{\|\hat{\mathbf{b}}\|}, \tag{2.29}$$

т.е. малые возмущения правой части задачи увеличиваются (или усиливаются) пропорционально числу обусловленности, поэтому иногда число обусловленности называют коэффициентом усиления шума. Задачи с большим числом обусловленности k матрицы A^TA принято называть *плохо обусловленными задачами*, а у задач с вырожденной матрицей A^TA число обусловленности $k = \infty$. Конечно, плохая обусловленность задачи это скорее качественная характеристика, потому-что нельзя указать универсальное пороговое значение числа обусловленности k при котором задача становится плохо обусловленной, хотя обычно в литературе говорят о $k > 1000$. Скорее, по мере увеличения k свойства решения задачи плавно ухудшаются, и в какой-то момент становится очевидно, что для получения разумного решения задачи следует применять специальный подход.

Чтобы понять как взаимодействовать с вырожденными и плохо обусловленными задачами перечислим необходимые свойства, которыми должно обладать решение задачи. Впервые их сформулировал французский математик Жак Адамар, назовём обратную задачу задачу $\theta = R[\mathbf{b}]$ *корректно поставленной*, если одновременно выполняются все следующие условия:

1. для всякого $\mathbf{b}$ существует решение θ,
2. решение θ единственно,
3. задача устойчива, т.е.

$$\forall\epsilon > 0, \exists\delta(\epsilon) > 0 : \rho_b(\mathbf{b}_1, \mathbf{b}_2) \le \delta(\epsilon) \Rightarrow \rho_\theta(\theta_1, \theta_2) \le \epsilon. \tag{2.30}$$

В противном случае задача называется *некорректно поставленной*. Здесь $R[\mathbf{b}]$ обозначает некоторый оператор, который вычисляет оценку параметров модели θ по заданным измерениям $\mathbf{b}$, $\rho_b(\mathbf{b}_1, \mathbf{b}_2)$ и $\rho_\theta(\theta_1, \theta_2)$ обозначают метрики соответствующих пространств. Например, в случае метода наименьших квадратов:

$$\begin{aligned} R[\mathbf{b}] &= (A^TA)^{-1}A^T\mathbf{b}, \\ \rho_b(\mathbf{b}_1, \mathbf{b}_2) &= \|\mathbf{b}_1 - \mathbf{b}_2\|, \\ \rho_\theta(\theta_1, \theta_2) &= \|\theta_1 - \theta_2\|. \end{aligned} \tag{2.31}$$

С первым требованием корректности задачи мы уже встречались, когда посмотрели на модель (2.1) как на уравнение относительно θ. Действительно, если вектор правой части $\mathbf{b}$ принадлежит образу линейного оператора A, то решение такого уравнения существует, а если не принадлежит, то решения не существует. Однако, вместо вектора $\mathbf{b}$ мы имеет дело с вектором $\hat{\mathbf{b}}$, отягощённым случайной погрешностью, и вектор $\hat{\mathbf{b}}$ в зависимости

от конкретной реализации погрешности может либо принадлежать образу линейного оператора A, либо нет. Такое поведение не имеет физического смысла, потому-что для эксперимента повторенного два раза в одном случае решение может существовать, а во втором — нет.

Второе требование тоже легко интерпретировать: искомые параметры модели являются физическими величинами, присущими конкретным объектам, и было бы очень странно, например, если бы один и тот же металлический шарик имел одновременно два различных значения массы.

Третье требование означает, что для близких результатов измерений должны получаться близкие значения оценок параметров модели. В противном случае, полученным оценкам может быть невозможно дать физическую интерпретацию, так как сильно различающеюся количественные характеристики могут приводить к различным качественными интерпретациям. Например, предположим, что у нас есть неустойчивый метод определения электропроводности какого-то вещества, тогда при разумной погрешности измерений соответствующий разброс получаемых значений оценки электропроводности может оказаться настолько велик, что будет невозможно сделать вывод диэлектрик перед нами или проводник.

Некорректные задачи часто встречаются в различных областях физики: от геофизики до астрономии, и заслуживают отдельного изучения. Подробности можно узнать в монографии А.Н. Тихонова (Тихонов и Арсенин 1986), признанного классика в исследовании некорректных задач. Однако, далее мы ограничимся рассмотрением случая вырожденных ($\det A^T A = 0$) и плохо обусловленных задач наименьших квадратов, чтобы на их примере продемонстрировать понятие *регуляризации* задачи.

Итак, пусть $\det A^T A = 0$, и нас по прежнему интересует решение задачи (2.8), тогда соответствующие нормальные уравнения (2.10) имеют множество решений, образующих линейное подпространство. *Нормальным относительно вектора $\theta_{\mathbf{0}}$ решением задачи (2.8)* будем называть $\hat{\theta}_{\mathbf{0}}$ такой что:

$$\hat{\theta}_{\mathbf{0}} = \arg\min_{\hat{\theta}} \|\hat{\theta} - \theta_{\mathbf{0}}\|^2, \tag{2.32}$$

$$\|A\hat{\theta} - \hat{\mathbf{b}}\|^2 = \min_{\theta} \|A\theta - \hat{\mathbf{b}}\|^2. \tag{2.33}$$

Просто *нормальным решением* будем называть нормальное решение относительно $\theta_{\mathbf{0}} = 0$. Очевидно, что нормальное решение $\hat{\theta}_{\mathbf{0}}$ единственно. Его физический смысл состоит в том, что мы предполагаем, что искомые параметры модели должны быть близки к $\theta_{\mathbf{0}}$, а отклонение от этой априорной оценки определяется исходя из нормы невязки модельного предсказания и измерений.

Далее покажем, почему вычисление нормального решение неустойчиво, строгое доказательство приведено в книге А.Н. Тихонова (см. Тихонов и Арсенин 1986). Итак, с помощью сингулярного разложения матрица A представима в виде произведения:

$$A = USV^T, \tag{2.34}$$

где U — ортогональная матрица размера n, V — ортогональная матрица размера m, S — прямоугольная матрица, все элементы которой равны нулю, кроме некоторых элементов, стоящих на её диагонали. Элементы, стоящие на диагонали матрицы S называются *сингулярными числами* и обозначаются σ_i, причём считается, что числа убывают с ростом индекса.[3] Собственные числа λ_i матрицы A^TA связаны с сингулярными числами матрицы A следующим образом:

$$\lambda_i = \sigma_i^2, \tag{2.35}$$

и, поскольку $\det A^TA = 0$, некоторые сингулярные числа σ_i равны нулю. Кроме того, введём следующие обозначения:

$$\begin{aligned} \mathbf{z} &\equiv V^T\theta, \\ \hat{\mathbf{u}} &\equiv U^T\hat{\mathbf{b}}. \end{aligned} \tag{2.36}$$

Теперь задача о поиске нормального решения запишется следующим образом:

$$\hat{\mathbf{z}}_\mathbf{0} = \arg\min_{\hat{\mathbf{z}}} \|\hat{\mathbf{z}} - \mathbf{z}_\mathbf{0}\|^2, \tag{2.37}$$

$$\|S\hat{\mathbf{z}} - \hat{\mathbf{u}}\|^2 = \min_{\mathbf{z}} \|S\mathbf{z} - \hat{\mathbf{u}}\|^2; \tag{2.38}$$

откуда благодаря диагональной форме матрицы S находится $\hat{\mathbf{z}}_\mathbf{0}$:

$$(\hat{z}_0)_i = \begin{cases} \dfrac{\hat{u}_i}{\sigma_i} & \sigma_i \neq 0 \\ (z_0)_i & \sigma_i = 0. \end{cases} \tag{2.39}$$

Понятно, что как только найден $\hat{\mathbf{z}}_\mathbf{0}$, то $\hat{\theta}_\mathbf{0} = V\hat{\mathbf{z}}_\mathbf{0}$ находится элементарно. Если σ_i находятся численно, как чаще всего и происходит, то, точно так же как и в случае с вычислением $\det A^TA$, результат будет отягощён ошибками округления. Это также можно интерпретировать, как внесение возмущения в матрицу A. Значит, на практике вместо вектора $\hat{\mathbf{z}}_\mathbf{0}$ будет вычислен некоторый вектор $\hat{\mathbf{z}}'_\mathbf{0}$:

$$(\hat{z}'_0)_i = \begin{cases} \dfrac{\hat{u}_i}{\hat{\sigma}_i} & \hat{\sigma}_i \neq 0, \\ (z_0)_i & \hat{\sigma}_i = 0; \end{cases} \tag{2.40}$$

где $\hat{\sigma}_i$ обозначают сингулярные значения, вычисленные с ошибкой округления, т.е. сингулярные значения возмущённой матрицы $\hat{A}$ (кстати, матрица $\hat{A}^T\hat{A}$ может оказаться уже не вырожденной, а просто плохо обусловленной). Видно, что когда для некоторого i сингулярное число $\sigma_i = 0$, а его возмущённая версия $\hat{\sigma}_i \neq 0$ (либо наоборот), расстояние между векторами $\hat{\mathbf{z}}_\mathbf{0}$ и $\hat{\mathbf{z}}'_\mathbf{0}$ может оказаться сколь угодно большим. Это показывает неустойчивость задачи нахождения нормального решения.

[3] Путем перестановки соответствующих столбцов, элементов, и строк матриц U, S, V^T можно добиться любого порядка следования диагональных элементов.

Процедура замены некорректной задачи на близкую по смыслу корректную задачу называется *регуляризацией*. Например, вместо поиска нормального решения вырожденной задачи мы могли бы искать решение $\hat{\theta}_\delta$, стремящееся к нормальному решению невозмущённой задачи $\hat{\theta}_{\mathbf{0}}$ по мере уменьшения возмущений, но не обязательно минимизирующее норму строго:

$$\|A\hat{\theta}_\delta - \hat{\mathbf{b}}\|^2 \leq \min_\theta \|A\theta - \hat{\mathbf{b}}\|^2 + 2\delta. \tag{2.41}$$

Подразумевается, что все возмущённые задачи для нас неразличимы на практике, значит и от идеального минимума можно отступить на характерную величину возмущения.

Такой подход называется *регуляризацией Тихонова*[4] или регуляризацией в второй норме, и приводит к следующей модификации задачи наименьших квадратов:

$$\hat{\theta}_\delta = \arg\min_\theta \left(\|A\theta - \hat{\mathbf{b}}\|^2 + \alpha \|\theta - \theta_{\mathbf{0}}\|^2 \right), \tag{2.42}$$

где α — некоторый параметр регуляризации. Нетрудно видеть, что решение выражается как

$$\hat{\theta}_\delta = \left(A^T A + \alpha I\right)^{-1} \left(A^T b + \alpha \theta_{\mathbf{0}}\right) \tag{2.43}$$

и при $\alpha \to 0$ решение стремиться к решению исходной задачи наименьших квадратов. При $\alpha \to \infty$, коэффициент обусловленности матрицы $A^T A + \alpha I$ приближается к единице, а $\hat{\theta}_\delta \to \theta_{\mathbf{0}}$.

На практике параметр α подбирается эмпирически. Одним из популярных способов поиска параметра α является метод L-кривой. Видно, что как только α зафиксирован, и, следовательно, $\hat{\theta}_\delta$ рассчитан, оба слагаемых в правой части выражения (2.42) могут быть рассчитаны по отдельности. Для построения L-кривой необходимо задать сетку величин α и для каждого значения поставить соответствующую точку на параметрической двумерной плоскости $\|A\theta_\delta(\alpha) - \hat{\mathbf{b}}\|^2$, $\|\theta_\delta(\alpha) - \theta_{\mathbf{0}}\|^2$. Получившийся график будет напоминать латинскую букву L, откуда и происходит его название. Оптимальное значение параметра α соответствует точке в месте максимального перегиба графика.

Возможный альтернативный подход — *регуляризация на основе усечённого сингулярного разложения*[5]. Вернёмся к выражению (2.40) и изменим его следующим образом:

$$(\hat{z}_\delta)_i = \begin{cases} \dfrac{\hat{u}_i}{\hat{\sigma}_i} & \hat{\sigma}_i \geq \sigma_0, \\ (z_0)_i & \hat{\sigma}_i < \sigma_0. \end{cases} \tag{2.44}$$

Фактически, подбирая порог σ_0 мы варьируем число обусловленности матрицы $\hat{A}^T \hat{A}$, которое составит:

$$k = \frac{\hat{\sigma}_1^2}{\sigma_0^2}. \tag{2.45}$$

[4] *ridge regression*
[5] *truncated SVD*

При достаточно большом значении σ_0 коэффициент обусловленности будет иметь умеренное значение, это значит, что вместо исходной плохо обусловленной задачи мы вычислим решение некоторой хорошо обусловленной задачи, близкой к исходной.

Запишем теперь решение задачи (2.42) в терминах сингулярного разложения, чтобы сравнить с (2.40) и (2.44):

$$(\hat{z}_\delta)_i = \frac{\hat{\sigma}_i}{\hat{\sigma}_i^2 + \alpha}\hat{u}_i + \frac{\alpha}{\hat{\sigma}_i^2 + \alpha}(z_0)_i\,. \tag{2.46}$$

Видно, что α и σ_0 имеют похожий смысл, и решения рассматриваемых подходов совпадают в двух предельных случаях: при $\alpha \to \infty$, $\sigma_0 \to \infty$ и при $\alpha \to 0$, $\sigma_0 \to 0$. В промежуточных условиях, в (2.46) априорное решение $\mathbf{z_0}$ подмешивается более гладко, чем в (2.44).

Отметим, однако, что подходы регуляризации предполагают включение в решение априорных знаний, т.е. знаний полученных до опыта или эксперимента. Видно, что в предельном случае, результат обработки данных от самих этих данных вовсе не зависит и определяется исключительно априорными предположениями о «правильном ответе». Если есть возможность улучшить плохую обусловленность задачи проведя дополнительные измерения при некоторых других, отличных от исследованных ранее, условиях, то такой вариант должен считаться предпочтительнее применения описанных здесь математических приёмов.

Поясним это на следующем примере. Введём матрицу $S^\dagger$, состоящую из диагонали:

$$\left(S^\dagger\right)_{ii} = \begin{cases} \sigma_i^{-1} & \sigma_i \geq \sigma_0, \\ 0 & \sigma_i < \sigma_0; \end{cases} \tag{2.47}$$

тогда оценка (2.44) запишется в следующем векторном виде:

$$\hat{\theta} = VS^\dagger U^T \hat{\mathbf{b}}. \tag{2.48}$$

Аналогично выражению (2.12) можно вычислить среднее значение оценки на основе усечённого сингулярного разложения:

$$\mathrm{E}\left[\hat{\theta}\right] = VS^\dagger SV^T\theta, \tag{2.49}$$

откуда видно, что теперь оценка $\hat{\theta}$ имеет смещение. Кроме того, можно вычислить матрицу ковариации оценки $\hat{\theta}$ аналогично (2.18):

$$\operatorname{cov}\hat{\theta} = \sigma^2 VS^\dagger\left(VS^\dagger\right)^T, \tag{2.50}$$

однако теперь эта величина характеризует только амплитуду разброса значений оценки $\hat{\theta}$ при проведении идентичных измерений. Видно, что этот разброс зависит от порога σ_0. Если нас интересует насколько оценка $\hat{\theta}$ отличается от истинных значений параметров θ, то необходимо учесть ещё и смещение:

$$\mathrm{E}\left[\|\hat{\theta} - \theta\|^2\right] = \sigma^2 \operatorname{tr} VS^\dagger\left(VS^\dagger\right)^T + \|\left(I - VS^\dagger SV^T\right)\theta\|^2, \tag{2.51}$$

причём два слагаемых находятся в противоположной зависимости через σ_0.

Глава 3

Анализ главных компонент

В главе 2 рассматривался метод наименьших квадратов, как способ оценки неизвестных параметров линейной модели. В разделе 2.4 использовалось сингулярное разложение матрицы. С сингулярным разложением матрицы ассоциируется *анализ (метод) главных компонент*[1] — одновременно мощный и изящный подход к анализу измерений на основе линейных моделей. В отличии от главы 2, преимущественно речь идёт об эмпирических моделях.

Итак, пусть $\mathbf{x} \in \mathbf{R}^n$ — случайный вектор с нулевым средним, т.е. $\mathrm{E}[\mathbf{x}] = 0$. Ковариационную матрицу вектора $\mathbf{x}$ будем обозначать Σ. Про матрицу Σ сразу известно, что она симметричная и положительно определённая, как и любая ковариационная матрица. Ковариационная матрица полагается неизвестной, и при необходимости оценивается на практике с использованием доступной выборки реализаций случайного вектора $\mathbf{x}$.

Требование о равенстве нулю среднего значения вектора не сильно ограничит наши рассуждения, обычно считается, что на практике, если известна выборка реализаций вектора $\mathbf{x}$ достаточного размера, то можно оценить среднее значение с удовлетворительной точностью, и вычесть оценку из каждой реализации.

3.1 Проекция с максимальной дисперсией

Зададимся вопросом, существует ли такой детерминированный вектор $\mathbf{q} \in \mathbf{R}^n$, чтобы проекция случайного вектора $\mathbf{x}$ на вектор $\mathbf{q}$, равная $(\mathbf{x}, \mathbf{q})$, имела бы максимально возможную дисперсию? Понятно, что эта проекция, т.е. скалярное произведение с участием случайного вектора — случайное число, которое имеет своё распределение, среднее и дисперсию. Не трудно заметить, что дисперсия может быть сделана сколь угодно большой за счёт умножения вектора $\mathbf{q}$ на произвольную константу, поэтому, чтобы вопрос о максимуме дисперсии имел смысл, необходимо ограничить норму вектора: $\|\mathbf{q}\|^2 = 1$.

[1] *principal component analysis (PCA)*

Отметим, что максимальная дисперсия интересует нас из соображений так называемого информационного содержания. Например, дифференциальная энтропия нормального распределения:

$$H = -\int p(x)\ln p(x)dx = \frac{1}{2}\sigma^2 \ln 2\pi e, \tag{3.1}$$

т.е. чем больше дисперсия, тем больше энтропия, и тем больше информации несёт реализация случайной величины.

Средняя проекция очевидно равна нулю:

$$\mathrm{E}\left[(\mathbf{x},\mathbf{q})\right] = (\mathrm{E}\left[\mathbf{x}\right],\mathbf{q}) = 0, \tag{3.2}$$

т.е. выражение для дисперсии проекции можно записать в следующем виде:

$$\mathrm{E}\left[(\mathbf{x},\mathbf{q})^2\right] = (\mathbf{q},\Sigma\mathbf{q}). \tag{3.3}$$

Воспользуемся подходом множителей Лагранжа для отыскания условного максимума этой функции относительно вектора $\mathbf{q}$. Нужно максимизировать следующую функцию

$$\Phi(\mathbf{q}) = (\mathbf{q},\Sigma\mathbf{q}) + \lambda(1 - \|\mathbf{q}\|^2), \tag{3.4}$$

где λ неизвестный множитель Лагранжа. Приравняв градиент целевой функции к нулю получим следующее уравнение:

$$\Sigma\mathbf{q} = \lambda\mathbf{q}. \tag{3.5}$$

Это задача на собственные значения и собственные вектора матрицы Σ. Известно, что её решением являются n собственных значений λ_k и соответствующие им собственные вектора $\mathbf{q}_k$. Заметим, что Σ положительно определённая матрица, следовательно все собственные значения $\lambda_k > 0$. Для удобства будем считать, что собственные значения λ_k пронумерованы в порядке убывания значения: λ_1 — максимальное собственное значение, а λ_n — минимальное.

Пусть $\mathbf{q}_k$ — собственный вектор матрицы Σ, отвечающий собственному значению λ_k, тогда дисперсия проекции на вектор $\mathbf{q_k}$ записывается следующим образом:

$$\mathrm{E}\left[(\mathbf{x},\mathbf{q}_k)^2\right] = (\mathbf{q}_k,\Sigma\mathbf{q}_k) = \lambda_k(\mathbf{q}_k,\mathbf{q}_k) = \lambda_k. \tag{3.6}$$

Откуда становится очевидным, что решением интересующей нас задачи является собственный вектор матрицы Σ, отвечающий максимальному собственному значению этой матрицы. Кроме того, собственное значение λ_k имеет смысл дисперсии проекции случайного вектора $\mathbf{x}$ на собственный вектор $\mathbf{q}_k$. Итак, собственный вектор $\mathbf{q}_1$ отвечающий максимальному собственному значению матрицы Σ называют *первым главным компонентом* вектора $\mathbf{x}$.

Попробуем обобщить задачу и построить такой ортогональный базис некоторой фиксированной размерности (т.е. задать линейное подпространство), чтобы проекция случайного вектора $\mathbf{x}$ на это линейное подпространство имело бы максимально возможную дисперсию в смысле суммы дисперсий проекций на отдельные орты. Решим задачу по индукции. Пусть Q — матрица, столбцы которой сформированы из ортонормированного базиса некоторого линейного подпространства. Выше мы доказали, что в случае одномерного подпространства единственный столбец этой матрицы — собственный вектор матрицы Σ, отвечающий максимальному собственному значению матрицы. Найдём максимум дисперсии проекции $(\mathbf{x}, \mathbf{q})$ среди всех векторов $\mathbf{q}$ с единичной нормой и ортогональных линейному подпространству, т.е. $Q^T\mathbf{q} = 0$.

Используя подход множителей Лагранжа, запишем целевую функцию:

$$\Phi(\mathbf{q}) = (\mathbf{q}, \Sigma\mathbf{q}) + \lambda(1 - \|\mathbf{q}\|^2) + (\mu, Q^T\mathbf{q}), \tag{3.7}$$

где λ — неизвестный множитель Лагранжа, а μ — целый вектор множителей Лагранжа, количество компонент которого равно размерности линейного подпространства. Приравняв градиент целевой функции к нулю получим следующее уравнение:

$$\Sigma\mathbf{q} - \lambda\mathbf{q} + \frac{1}{2}Q\mu = 0. \tag{3.8}$$

Покажем, что третье слагаемое на самом деле всегда равно нулю. Для этого домножим (3.8) слева на Q^T:

$$Q^T\Sigma\mathbf{q} - \lambda Q^T\mathbf{q} + \frac{1}{2}Q^TQ\mu = Q^T\Sigma\mathbf{q} + \frac{1}{2}\mu = 0. \tag{3.9}$$

Откуда видно, что вектор μ всегда равен нулю:

$$\mu = -2Q^T\Sigma\mathbf{q} = -2(\Sigma Q)^T\mathbf{q} = -2\Lambda Q^T\mathbf{q} = 0, \tag{3.10}$$

здесь Λ — диагональная матрица, составленная из собственных значений матрицы Σ, отвечающих соответствующим колонкам Q.

Итак, задача сводится к предыдущей:

$$\Sigma\mathbf{q} = \lambda\mathbf{q}, \tag{3.11}$$

однако на этот раз в качестве ответа нужно выбрать собственное значение λ максимальное среди всех не использованных собственных значений λ_k, т.е. таких, чьи собственные вектора q_k не содержатся в Q. Поэтому, собственный вектор $\mathbf{q}_2$ отвечающий второму по величине собственному значению матрицы Σ называют *вторым главным компонентом*, по аналогии все собственные вектора матрицы Σ называют главными компонентами. Аналогично, собственный вектор $\mathbf{q}_n$ отвечающий минимальному собственному значению матрицы Σ иногда называют *первым побочным компонентом*.

Во-первых, нетрудно видеть, что координаты проекции вектора $\mathbf{x}$ на базис Q некореллированны, это следствие ортогональности базиса Q:

$$\mathrm{E}\left[(\mathbf{x}, \mathbf{q}_k)(\mathbf{x}, \mathbf{q}_l)\right] = (\mathbf{q}_k, \Sigma\mathbf{q}_l) = \lambda_l(\mathbf{q}_k, \mathbf{q}_l). \tag{3.12}$$

Это свойство зачастую используется в рамках подходов машинного обучения, когда метод анализа главных компонент используется в качестве начальных шагов при подготовке данных.

Во-вторых, анализ главных компонент часто используется в качестве метода *понижения размерности* данных, так как для любой заданной размерности подпространства позволяет найти базис, сохраняющий максимум информационного содержания случайного вектора $\mathbf{x}$. Понижение размерности требуется, например, для визуализации данных; либо для последующего применения к данным методов, эффективность которых существенно зависит от размерности входных данных; либо в случаях, когда размерность входных данных сильно больше размера выборки, т.е. числа измерений.

В-третьих, метод можно рассматривать как способ фильтрации шума в данных. Задав размерность подпространства, мы считаем, что главные компоненты отвечают за информационное содержание данных, т.е. задают возможное разнообразие измерений, а побочные компоненты отвечают за шум, т.е. задают флуктуации измерений. Таким образом, спроектировав вектор $\mathbf{x}$ в линейное подпространство меньшей размерности, можно отфильтровать случайный шум, сохранив важную информацию. Заметим, что, аналогично методу регуляризации на основе усечённого сингулярного разложения из раздела 2.4, выбирая размерность подпространства мы балансируем между сохранением информации, содержащейся в данных, и уровнем шума.

3.2 Проекция с минимальной ошибкой

Пусть $\mathbf{x} \in \mathbf{R}^n$ всё тот же случайный вектор, на этот раз мы хотим найти такой вектор $\mathbf{q}$, проекция на который минимизирует ошибку проекции в смысле квадрата нормы разности:

$$R(\mathbf{x}) = \|(\mathbf{x}, \mathbf{q})\mathbf{q} - \mathbf{x}\|^2. \tag{3.13}$$

Иными словами, нам хотелось бы заменить случайный вектор на одно случайное число, по возможности потеряв при этом минимум информации, однако теперь мы формулируем эту задачу по другому.

Легко заметить, что средняя ошибка выражается как:

$$\mathrm{E}\left[\|(\mathbf{x}, \mathbf{q})\mathbf{q} - \mathbf{x}\|^2\right] = \operatorname{tr}\Sigma - (\mathbf{q}, \Sigma\mathbf{q}). \tag{3.14}$$

Используя подход множителей Лагранжа для того, чтобы ограничить норму $\mathbf{q}$, запишем целевую функцию в следующем виде:

$$\Phi(\mathbf{q}) = \operatorname{tr}\Sigma - (\mathbf{q}, \Sigma\mathbf{q}) + \lambda(\|\mathbf{q}\|^2 - 1), \tag{3.15}$$

откуда после взятия градиента вновь получается задача на собственные значения матрицы Σ:

$$\Sigma\mathbf{q} = \lambda\mathbf{q}. \tag{3.16}$$

Средняя ошибка проекции выражается через собственное значение матрицы λ_k и собственный вектор $\mathbf{q}_k$ следующим образом:

$$\mathrm{E}\left[\|(\mathbf{x},\mathbf{q}_k)\mathbf{q}_k-\mathbf{x}\|^2\right]=\operatorname{tr}\Sigma-(\mathbf{q}_k,\Sigma\mathbf{q}_k)=\operatorname{tr}\Sigma-\lambda_k=\sum_{j=1}^{n}\lambda_j-\lambda_k. \quad (3.17)$$

Т.е. из суммы положительных собственных значений предлагается вычесть одно слагаемое так, чтобы результат был минимально возможным. Очевидно, что в качестве решения нужно брать максимальное собственное значение λ_1 и соответствующий ему собственный вектор $\mathbf{q}_1$, т.е. первый главный компонент является решением задачи.

Аналогичным образом задача решается для проекции на многомерное подпространство. Пусть Q — матрица, m колонок которой составлены из векторов ортонормированного базиса некоторого линейного подпространства, нас интересует такой базис, чтобы средняя ошибка проекции вектора $\mathbf{x}$ в соответствующее линейное подпространство, вычисленная следующем образом

$$\mathrm{E}\left[\|QQ^T\mathbf{x}-\mathbf{x}\|^2\right]=\operatorname{tr}\Sigma-\operatorname{tr}Q^T\Sigma Q, \quad (3.18)$$

была минимально возможной.

На этот раз придётся ввести целую матрицу множителей Лагранжа $H=H^T$ и записать целевую функцию в следующем виде:

$$\Phi(Q)=\operatorname{tr}\Sigma-\operatorname{tr}Q^T\Sigma Q+\operatorname{tr}H\left(Q^TQ-I\right). \quad (3.19)$$

Нетрудно взять производную от этого выражения по каждому из компонент искомой матрицы Q и получить следующее уравнение для условного экстремума:

$$\Sigma Q=QH, \quad (3.20)$$

используя симметрию H и разложение $H=V\Lambda V^T$, где V - ортогональная матрица, а Λ - диагональная, перепишем уравнение в следующем виде:

$$\Sigma(QV)=(QV)\Lambda, \quad (3.21)$$

произведение QV говорит нам о том, что искомый базис можно поворачивать и отражать как угодно, и на результат задачи (т.е. минимальность средней ошибки) такая операция не повлияет. Это достаточно очевидное и ожидаемое свойство, поэтому положим $V=I$:

$$\Sigma Q=Q\Lambda, \quad (3.22)$$

Так как матрица Λ диагональная, то для каждого столбца $\mathbf{q}$ матрицы Q можно записать отдельно знакомое нам уравнение:

$$\Sigma\mathbf{q}=\lambda\mathbf{q}. \quad (3.23)$$

Значение средней ошибки проекции равно:

$$\mathrm{E}\left[\|QQ^T\mathbf{x}-\mathbf{x}\|^2\right]=\operatorname{tr}\Sigma-\operatorname{tr}Q^T\Sigma Q=\sum_{j=1}^{n}\lambda_j-\sum_{k:\ \mathbf{q}_k\in Q}\lambda_k. \quad (3.24)$$

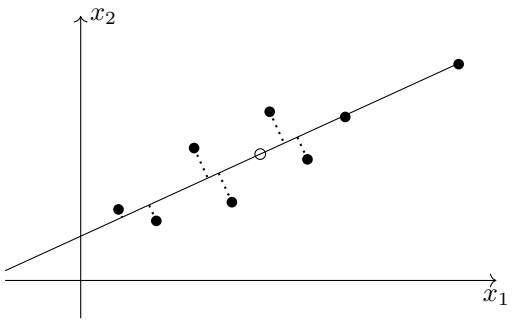

Рис. 3.1: Применение анализа главных компонент для проекции выборки двумерных векторов $\mathbf{x}_i$ на одномерное линейное подпространство. Точками показаны положения выборки $\mathbf{x}_i$; кружком показано положение выборочного среднего; сплошной линией, проходящей через положение выборочного среднего, показано направление первого главного компонента $\mathbf{q}_1$; пунктирные линии показывают минимизируемое расстояние между измерениями и проекцией на первый главный компонент. По сравнению с методом наименьших квадратов на Рис. 2.1, минимизируемые расстояния измеряются не параллельно вертикальной оси, а перпендикулярно к модельной прямой.

Понятно, что решение задачи состоит в том, чтобы взять m максимальных по значению собственных значений, а соответствующие им собственные вектора (т.е. главные компоненты) использовать в качестве базиса искомого линейного подпространства. На Рис. 3.1 приведён графический пример применения анализа главных компонент к выборке точек на плоскости.

Мы ещё раз подтвердили свойство главных компонент: сохранение информации при понижении размерности. Понятно, что вектор после проекции на линейное подпространство меньшей размерности остаётся похож на самого себя с наименьшей ошибкой в смысле квадрата нормы невязки.

3.3 Сингулярное разложение и анализ главных компонент

На практике обычно известна некоторая конечная выборка из N реализаций случайного вектора $\mathbf{x}$, поэтому нам интересно, как вычислять главные компоненты, и проекции реализаций случайного вектора на линейное подпространство меньшей размерности.

Наивный способ мог вы состоять в том, чтобы сначала найти выборочную оценку матрицу ковариации этого вектора:

$$\hat{\Sigma} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i \otimes \mathbf{x}_i, \tag{3.25}$$

где $\otimes$ обозначает внешнее векторное произведение, а затем вычислить собственные значения и собственные вектора этой матрицы тем или иным алгоритмом.

Однако, на практике предпочитают другой способ, основанный на вычислении сингулярного разложения. Назовём *матрицей данных* или *матрицей измерений* матрицу X, строки которой состоят из доступных реализацией случайного вектора $\mathbf{x}$. Таким образом X имеет N строк и n столбцов. Тогда оценку ковариации (3.25) можно записать в матричном виде:

$$\hat{\Sigma} = \frac{1}{N} X^T X \tag{3.26}$$

Запишем сингулярное разложение матрицы X:

$$X = USV^T, \tag{3.27}$$

где U — ортогональная матрица размера N, V — ортогональная матрица размера n, S — прямоугольная матрица, все элементы которой равны нулю, кроме элементов, стоящих на её диагонали. Заметим, что

$$\hat{\Sigma} = \frac{1}{N} X^T X = \frac{1}{N} V S^2 V^T, \tag{3.28}$$

откуда видно, что матрица V состоит из колонок, соответствующих главным компонентам, а диагональная матрица S^2 состоит из соответствующих им собственных значений. Значит матрица данных, записанная в базисе главных компонент, выражается следующим образом:

$$XV = USV^T V = US. \tag{3.29}$$

Напомним, что все элементы матрицы S ниже строки n равны нулю, значит для вычисления произведения US достаточно вычислить и хранить в памяти только первые n колонок матрицы U. Алгоритмы вычисления сингулярного разложения, реализованные в популярных библиотеках, как правило позволяют использовать эту оптимизацию.

Глава 4

Численные методы оптимизации

Задачи оптимизации, т.е. задачи поиска минимума или максимума некоторой функции при определённых условиях, играют важную роль в обработке и анализе данных. Действительно, в главе 2 мы уже столкнулись с необходимость поиска минимума квадрата нормы невязки в методе наименьших квадратов, в главе 3 оказалось, что главные компоненты образуют базис, минимизирующий ошибку проекции данных на этот базис и одновременно максимизирующий дисперсию проекции. В главе 5 будет рассмотрен метод максимального правдоподобия, из самого названия которого понятно, что речь идёт о максимизации некоторой функции. Методы машинного обучения, например, нейронные сети (см. раздел 8.5) или ансамбли решающих деревьев (см. раздел 8.1) основаны на задачах оптимизации.

К сожалению, далеко не всегда удаётся решить задачу оптимизации аналитически и свести её к вычислениям по готовой формуле, как это происходит в линейном методе наименьших квадратов (см. главу 2) или анализе главных компонент (см. главу 3). В таком случае необходимо использовать численные методы оптимизации. На первый взгляд может показаться не очевидным, но численные методы оптимизации могут оказаться предпочтительными даже в тех случаях, когда задача оптимизации решается аналитически. Например, для решения задачи линейных наименьших квадратов с матрицами задачи специального вида часто применяется метод сопряжённых градиентов (см. раздел 4.2.2).

Основная сложность состоит в том, что универсального эффективного алгоритма численной оптимизации не существует. Как правило, библиотеки методов оптимизации предоставляют целый набор разных реализованных алгоритмов на выбор. Интересно, что типичные наборы алгоритмов оптимизации из библиотеки общего назначения и из библиотеки для работы с нейронными сетями обычно имеют мало общих методов. Такое разнообразие обусловлено тем, что задачи оптимизации бывают слишком разными.

Процедуру выбора конкретного алгоритма оптимизации можно разделить на два этапа: во-первых, выбор алгоритма, который теоретически способен решить конкретную задачу; во-вторых, выбор алгоритма, который справляется с решением этой задачи наиболее эффективным образом, где под эффективностью может пониматься скорость работы, точность результата, и т.п. Классификации задач оптимизации посвящён раздел 4.1, а алгоритмы оптимизации рассматриваются в разделе 4.2.

4.1 Задачи оптимизации

Задачей минимизации будем называть следующую задачу:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in X} f(\mathbf{x}), \tag{4.1}$$

где $f(\mathbf{x})$ называют *целевой функцией*[1], а X — называют *допустимым множеством*[2]. Сразу заметим, что *задача максимизации* сводится к задаче минимизации с помощью приписывания знака минус перед целевой функцией, поэтому принято говорить сразу о *задачах оптимизации*. Кроме того, в задачах оптимизации, возникающих при обработке и анализе данных, положение минимума $\mathbf{x}^*$ чаще всего имеет больший физический смысл, чем значение целевой функции в этой точке $f(\mathbf{x}^*)$. Действительно, мы видели в линейном методе наименьших квадратов, что положение минимума целевой функции использовалось нами как оценка неизвестных параметров модели, а значение целевой функции в точке минимума имело смысл некоторой точности. Понятно, что чаще всего нас интересуют именно оценки значений параметров той или иной модели. По этой же причине аддитивные константы в целевой функции часто игнорируются.

Напомним, что $\mathbf{x}^*$ называется точкой *глобального минимума* если

$$f(\mathbf{x}^*) \leq f(\mathbf{x}), \quad \forall \mathbf{x} \in X, \tag{4.2}$$

если же условие выполняется в строгой форме, то говорят о *строгом глобальном минимуме*. Если условие выполняется лишь для некоторой окрестности точки $\mathbf{x}^*$, то говорят, что имеет место *локальный минимум*. Неограниченная снизу на множестве X функция $f(\mathbf{x})$ не имеет глобального минимума, но может иметь один или несколько локальных минимумов; ограниченная снизу функция $f(\mathbf{x})$ имеет глобальный минимум (это утверждение иногда известно под именем теоремы Вейерштрасса), но может иметь и несколько локальных минимумов.

Если допустимое множество X совпадает со всем пространством $\mathbf{R}^n$, как часто и случается, то в таком случае задача оптимизации называется *безусловной задачей оптимизации* или *неограниченной задачей оптимизации*[3]:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbf{R}^n} f(\mathbf{x}), \tag{4.3}$$

[1] *cost function*
[2] *feasible set*
[3] *unconstrained optimization problem*

иначе — задача называется *условной задачей оптимизации* или *ограниченной задачей оптимизации*[4]. В этом случае допустимое множество X чаще всего задаётся с помощью ограничений в виде равенств и неравенств, тогда такая задача называется *задачей математического программирования*:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbf{R}^n} f(\mathbf{x}), \tag{4.4}$$

$$g_i(\mathbf{x}) = 0 \quad i \in \mathcal{E}, \tag{4.5}$$

$$g_i(\mathbf{x}) \geq 0 \quad i \in \mathcal{I}. \tag{4.6}$$

Функции $g_i(\mathbf{x})$ называются ограничениями. Они и задают допустимое множество X. Понятно, что множества $\mathcal{E}$ и $\mathcal{I}$ могут быть пустыми, таким образом можно сформулировать задачи с ограничениями равенствами и ограничениями неравенствами. Более-менее очевидно, что для задачи ограниченной оптимизации в виде задачи математического программирования проще предложить какие-то алгоритмы поиска минимума, чем если бы допустимое множество X было бы задано некоторым более замысловатым образом.

Если допустимое множество X является дискретным множеством, либо $f(\mathbf{x})$ является дискретной функцией, то такая задача оптимизации называется *задачей дискретной оптимизации*. Задачи дискретной оптимизации чаще всего являются NP-полными задачами (см. раздел 1.3.1). Примером таких задач является задача о рюкзаке (1.8)–(1.9). К счастью, в физике такие задачи встречаются скорее в виде исключений.

Важно различать точки локального минимума функции $f(\mathbf{x})$ и *критические точки* (или *стационарные точки*) функции $f(\mathbf{x})$, потому-что большинство алгоритмов численной оптимизации находят именно критические точки первого или второго порядка, в то время как нас интересует вообще говоря глобальный минимум. Определения в виде (4.2) вообще плохо алгоритмизируются, хотя бы потому, что проверка выполнения условия для бесконечного набора точек окрестности $\mathbf{x}^*$ занимает бесконечное количество вычислительного времени.

Итак, *критической точкой первого порядка* неограниченной задачи оптимизации (4.3) называется точка $\mathbf{x}^*$ такая, что градиент целевой функции в этой точке равен нулю:

$$\nabla f(\mathbf{x}^*) = 0. \tag{4.7}$$

Необходимое условие оптимальности первого рода можно сформулировать следующим образом: если $\mathbf{x}^*$ локальный минимум $f(\mathbf{x})$, то $\mathbf{x}^*$ критическая точка первого порядка.

Критической точкой второго порядка неограниченной задачи оптимизации (4.3) называется критическая точка первого порядка $\mathbf{x}^*$ такая, что вычисленная в данной точке матрица вторых производных целевой функции

$$(H)_{ij} \equiv \left.\frac{\partial^2 f}{\partial x_i \partial x_j}\right|_{\mathbf{x}^*}, \tag{4.8}$$

[4] *constrained optimization problem*

известная как *матрица Гессе* или *гессиан*, неотрицательно определена:

$$(\mathbf{h}, H\mathbf{h}) \geq 0, \quad \forall \mathbf{h} \in \mathbf{R}^n. \tag{4.9}$$

Необходимое условие оптимальности второго рода может быть сформулировано аналогично: если $\mathbf{x}^*$ локальный минимум $f(\mathbf{x})$, то $\mathbf{x}^*$ критическая точка второго порядка. Иными словами, все точки минимума являются критическими точками, но не все критические точки являются точками локального минимума. Например, для функции $f(x) = x^3$, неограниченной снизу, точка $x^* = 0$ очевидно критическая точка, но не положение локального минимума.

Понятно, что для задач ограниченной оптимизации требуются свои определения. *Критической точкой первого порядка* для задачи математического программирования (4.4)–(4.6) называется точка $\mathbf{x}^*$, если существует такой вектор $\mathbf{y}^*$, при котором выполняются следующие условия известные под названием *условий Каруша-Куна-Такера* или просто *условий Куна-Такера*:

$$\nabla f(\mathbf{x}^*) - \sum_{i \in \mathcal{E} \cup \mathcal{I}} y_i^* \nabla g_i(\mathbf{x}^*) = 0, \tag{4.10}$$

$$g_i(\mathbf{x}^*) = 0 \quad i \in \mathcal{E}, \tag{4.11}$$

$$g_i(\mathbf{x}^*) \geq 0 \quad \text{и} \quad y_i^* \geq 0 \quad i \in \mathcal{I}, \tag{4.12}$$

$$g_i(\mathbf{x}^*) y_i^* = 0 \quad i \in \mathcal{I}. \tag{4.13}$$

Условия (4.10)–(4.13) представляют собой необходимое условие оптимальности первого рода для задачи (4.4)–(4.6).[5]

Условия критической точки первого рода могут иметь следующую интересную геометрическую интерпретацию. Назовём *активными ограничениями* ограничения $g_i(\mathbf{x}) = 0$ для $i \in \mathcal{E} \cup \mathcal{I}$, понятно, что ограничения равенства всегда активные. Иначе, ограничения называются *пассивными ограничениями*. Тогда из условий (4.10) и (4.13) видно, что $\nabla f(\mathbf{x}^*)$ является линейной комбинацией векторов $\nabla g_i(\mathbf{x}^*)$ для всех активных ограничений. Пусть матрица $N \in \mathbf{R}^{n \times m}$ состоит из колонок, образующих базис

[5]Вообще говоря, формально условия Каруша-Куна-Такера (4.10)–(4.13) сформулированы только для так называемых *регулярных задач* ограниченной оптимизации. Приведём пример нерегулярной задачи математического программирования (Сухарев, Тимохов и Федоров 2008):

$$\mathbf{x}^* = \arg \min_{\mathbf{x} \in \mathbf{R}^2} x_1,$$
$$x_1^3 - x_2 \geq 0,$$
$$x_1^3 + x_2 \geq 0,$$
$$1 - x_1^2 - x_2^2 \geq 0.$$

Не составит большого труда убедиться в том, что $\mathbf{x}^* = 0$ — решение такой задачи, однако условия Каруша-Куна-Такера для $\mathbf{x}^*$ очевидно не выполняются. Для нас важно то, что большинство известных алгоритмов численного решения задачи оптимизации (4.4)–(4.6) сходятся в точки, где выполняются эти условия. Поэтому с нашей точки зрения проблема именно в сложности нерегулярных задач, а вовсе не в том, что условия Каруша-Куна-Такера недостаточно общо сформулированы.

линейного подпространства, ортогонального ко всем градиентам активных ограничений, тогда проекция вектора градиента целевой функции на это подпространство, называемое *нуль-пространством*:

$$N^T \nabla f(\mathbf{x}^*) = 0, \tag{4.14}$$

т.е. градиент целевой функции равен нулю в том подпространстве, где ограничения «не действуют».

Критической точкой второго порядка в слабом смысле[6] для математического программирования (4.4)–(4.6) называется критическая точка первого порядка $\mathbf{x}^*$, такая, что проекция матрицы вторых производных на нуль-пространство неотрицательно определена:

$$(\mathbf{h}, N^T H N \mathbf{h}) \geq 0, \quad \forall \mathbf{h} \in \mathbf{R}^m. \tag{4.15}$$

Кроме того, допустимое множество X может быть выпуклым множеством. Напомним, что выпуклым множеством называется такое множество, что

$$\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2 \in X, \quad \forall \mathbf{x}_1, \mathbf{x}_2 \in X, \quad \lambda \in [0,1]. \tag{4.16}$$

Некоторые полезные примеры выпуклых множеств:

- $\mathbf{R}^n$;
- $\|\mathbf{x} - \mathbf{x}_0\| \leq \Delta$;
- $A\mathbf{x} \leq \mathbf{b}$, где под сравнением векторов понимается система поэлементных неравенств.

Целевая функция $f(\mathbf{x})$ в свою очередь тоже может быть выпуклой. Напомним, что выпуклой называется функция такая, что

$$f(\lambda \mathbf{x}_1 + (1-\lambda)\mathbf{x}_2) \leq \lambda f(\mathbf{x}_1) + (1-\lambda) f(\mathbf{x}_2), \quad \forall \mathbf{x}_1, \mathbf{x}_2 \in X, \lambda \in [0,1]. \tag{4.17}$$

Примерами выпуклых функций являются:

- x^2,
- $\|\mathbf{x}\|^2$,
- $\|A\mathbf{x} - \mathbf{b}\|^2$.

Задача оптимизации называется *выпуклой задачей оптимизации*, если $f(\mathbf{x})$ и X выпуклые. Напомним, что $\mathbf{R}^n$ выпуклое, значит задача безусловной оптимизации — выпуклая задача, когда $f(\mathbf{x})$ выпуклая. Можно доказать важную теорему о том, что выпуклая задача оптимизации имеет всего одну критическую точку, которая является положением глобального минимума целевой функции. Иными словами выпуклая задача оптимизации

[6] Определения критической точки второго порядка в сильном смысле мы не приводим ради краткости.

всегда имеет единственное решение, а достаточным условием оптимальности выпуклой функции в точке $\mathbf{x}^*$ является $\nabla f(\mathbf{x}^*) = 0$.

Некоторые задачи с конкретным видом целевой функции имеют свои собственные названия и специализированные алгоритмы решения.

Например, *задача линейного программирования* — это задача о поиске условного максимума линейной функции:

$$\mathbf{x}^* = \arg\max_{\mathbf{x}\in\mathbf{R}^n}(\mathbf{c},\mathbf{x}), \tag{4.18}$$

$$A\mathbf{x} \le \mathbf{b}, \tag{4.19}$$

где сравнение двух векторов $A\mathbf{x}$ и $\mathbf{b}$ понимается как поэлементное сравнение. Понятно, что если бы не ограничение (4.19), то задача поиска максимума неограниченной сверху функции $(\mathbf{c},\mathbf{x})$ не имела бы смысла.

Задача квадратичного программирования — это задача о поиске минимума квадратичной формы:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbf{R}^n}\left((\mathbf{c},\mathbf{x}) + \frac{1}{2}(\mathbf{x}, H\mathbf{x})\right). \tag{4.20}$$

Она имеет как самостоятельный интерес, например в случае метода наименьших квадратов (2.8), так и является вспомогательной задачей для некоторых численных методов оптимизации. Например, алгоритмы Ньютона и Левенберга–Марквандa (см. раздел. 4.2.1) предполагают, что задача (4.20) решается на каждом шаге.

Несмотря на то, что выражения для точки минимума этой функции $\mathbf{x}^*$ можно выписать аналитически:

$$\mathbf{x}^* = -H^{-1}\mathbf{c}, \tag{4.21}$$

существуют альтернативные численные методы решения этой задачи, такие, например, как метод сопряжённых градиентов (см. раздел. 4.2.2), одним из преимуществ которого является отсутствие необходимости обращать матрицу H.

Кстати, к задаче квадратичного программирования могут быть добавлены ограничения, следующие из физического смысла поставленной задачи. Например, неотрицательная задача наименьших квадратов предполагает, что искомые параметры физической модели не могут иметь отрицательные значения по своему физическому смыслу:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbf{R}^n}\|A\mathbf{x} - \hat{\mathbf{b}}\|^2, \tag{4.22}$$

$$\mathbf{x} \ge \mathbf{0}, \tag{4.23}$$

где сравнение вектора $\mathbf{x}$ с нулём снова понимается поэлементно. К сожалению, оценка параметров линейной модели, полученная как решение неотрицательной задачи наименьших квадратов, не обладает такими полезными свойствами как, например, несмещённость, однако обладает другим серьёзным свойством — физичностью получаемых значений параметров.

4.1.1 Задача нелинейных наименьших квадратов

Другим интересным частным случаем является *задача нелинейных наименьших квадратов*, попытка обобщения метода наименьших квадратов (см. главу 2) на случай нелинейных моделей:

$$\mathbf{x}^* = \arg\min_{\mathbf{x}\in\mathbf{R}^n} \frac{1}{2}\|\phi(\mathbf{x}) - \hat{\mathbf{b}}\|^2. \tag{4.24}$$

Отметим сразу, что в отличии от линейной задачи наименьших квадратов, оценки параметров нелинейных наименьших квадратов не обладают всеми теми свойствами, которыми обладает их линейный аналог. Кроме того, адекватная оценка ошибок оценок параметров может оказаться затруднена, особенно в случае сильно нелинейных задач. Тем не менее, подход нелинейных наименьших квадратов применяется на практике как для моделей имеющих физическое происхождение, так и для эмпирических моделей: например, решение задачи регрессии с помощью нейронных сетей (см. раздел 8.5) по-сути сводится к задаче (4.24).

Для оценки ошибок оценки параметров нелинейной модели методом наименьших квадратов воспользуемся приближением малых ошибок. Введём матрицу Якоби (или якобиан):

$$(J)_{ij}(\mathbf{x}) \equiv \left.\frac{\phi_i}{x_j}\right|_{\mathbf{x}}, \tag{4.25}$$

это матрица первых производных вектор-функции $\phi(\mathbf{x})$. Заметим, что в случае линейных наименьших квадратов матрица задачи A совпадает с якобианом J. Тогда по аналогии с формулой (2.18) получим следующее выражение:

$$\operatorname{cov}\mathbf{x}^* = \frac{\|\hat{\epsilon}\|^2}{n-m}\left(J(\mathbf{x}^*)^T J(\mathbf{x}^*)\right)^{-1}, \tag{4.26}$$

где

$$\hat{\epsilon} = \hat{\mathbf{b}} - \phi(\mathbf{x}^*). \tag{4.27}$$

Зачастую, ошибки определения параметров далее используются для проверки той или иной гипотезы (см. главу 7), например, нас интересует можно ли считать верным теоретически предсказанное значение только что найденного параметра. В таком случае важно, чтобы распределение $\mathbf{x}^*$ оставалось нормальным в ответ на нормальные ошибки $\hat{\mathbf{b}}$. В методе линейных наименьших квадратов это требование выполняется всегда автоматически.

В методе нелинейных наименьших квадратов выражение (4.26) справедливо лишь в случае малых ошибок, поэтому на практике следует соблюдать осторожность. На рисунке 4.1 демонстрируется ограниченность приближения малых ошибок. Видно, что если размер окрестности точки $\mathbf{x}^*$, где модельная функция $\phi(\mathbf{x})$ ведёт себя линейно, превышает ошибку параметров, то формула (4.26) приводит к верным результатам. Иначе, формула позволяет формально рассчитать оценку ошибки параметров, которая может

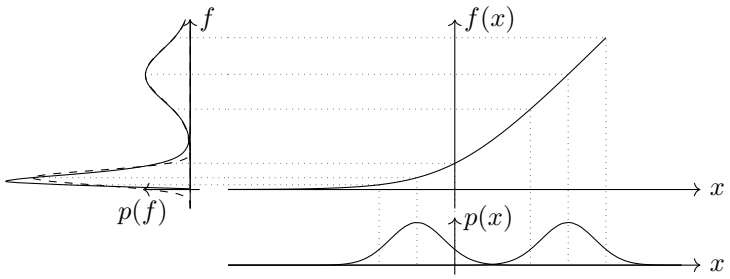

Рис. 4.1: Показан график некоторой зависимости $y = f(x)$, а так же плотности вероятности случайных величин x, y для двух случаев. Штриховой линией на левом графике показаны нормальные распределения с параметрами вычисленными в приближении малых ошибок. В первом случае, распределение x нормальное, а распределение y сильно искажено из-за нелинейности зависимости в данной области. Видно, что мода распределения отличается от среднего. Кроме того, нормальное распределение предсказывает ненулевую вероятность наблюдения значений $y < 0$, которые не могут существовать по построению функции $f(x)$. Во втором случае распределение y близко к нормальному.

быть лучше, чем отсутствие какой-либо оценки ошибки параметров, но не сгодится для рассуждений в терминах доверительных интервалов.

По аналогии с линейной задачей наименьших квадратов, число обусловленности матрицы Якоби J отвечает за обусловленность задачи, и формально обусловленность задачи может быть разной при разных значениях параметров $\mathbf{x}$. В случае плохой обусловленности задачи, численный поиск минимума становится затруднён, потому-что произведение $J^T J$ имеет смысл аналогичный матрице Гессе H в выражении (4.9) и становится трудно отличать критические точки второго рода от седловых точек.

4.2 Алгоритмы оптимизации

Когда говорят о численных методах оптимизации, в большинстве случаев подразумеваются итеративные алгоритмы оптимизации, т.е. алгоритмы, составленные из потенциально бесконечного числа одинаковых последовательных шагов, где на каждом шаге вычисляется улучшенный кандидат в решение задачи $\mathbf{x}^{(k+1)} = \mathbf{S}_k \left[\mathbf{x}^{(k)}\right]$ на основе известного кандидата в решение задачи $\mathbf{x}^{(k)}$. На следующем шаге операция повторяется, и вычисляется следующий улучшенный кандидат $\mathbf{S}_{k+1} \left[\mathbf{x}^{(k+1)}\right]$.

Вычисления, производимые на каждом шаге могут быть строго детерминированными, тогда алгоритм называют *детерминированным*, либо включать в себя элемент случайности, тогда алгоритм будет называться *стохастическим алгоритмом оптимизации* (см. раздел 4.4).

В качестве простого примера итеративного алгоритма оптимизации назовём *метод градиентного спуска*, в котором вычисления на каждом шаге производятся по следующему правилу:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \nabla f(\mathbf{x}^{(k)}), \tag{4.28}$$

где $\nabla f(\mathbf{x}^{(k)})$ обозначает вычисленный вектор градиента целевой функции в точке $\mathbf{x}^{(k)}$, а $\alpha_k > 0$ некоторый параметр алгоритма, обычно убывающий с ростом k.

Алгоритм шага $\mathbf{S}_k\left[\mathbf{x}^{(k)}\right]$ выбирается таким образом, чтобы можно было доказать сходимость получаемой последовательности через конечное или бесконечное число шагов. А в случае стохастических алгоритмов оптимизации — сходимость по вероятности или сходимость по распределению. В большинстве случаев, рассматриваемая последовательность сходится (или, как говорят, алгоритм оптимизации сходится) к критической точке первого рода или критической точке второго рода той или иной задачи оптимизации.

Таким образом, можно выделить несколько признаков, по которым классифицируются алгоритмы оптимизации. Во-первых, род критической точки, в которую сходится алгоритм. Например, если целевая функция выпуклая, то достаточно использовать алгоритм, сходящийся к критической точке первого рода.

Во-вторых, вид целевой функции, с которой алгоритм умеет работать. Более специализированные алгоритмы как правило лучше справляются с частными задачами оптимизации, такими, например, как задача квадратичного программирования (4.20), или, например, задача нелинейных наименьших квадратов (4.24). Специализированные алгоритмы, которые справлялись хуже алгоритмов оптимизации общего назначения, естественным образом оказались преданы забвению.

В-третьих, количество задействованной информации на каждом шаге. Если алгоритм оптимизации требует для работы возможность вычисления целевой функции $f(\mathbf{x})$ в любой точке, то он называется *алгоритмом нулевого порядка*, если дополнительно требуется возможность вычисления градиента $\nabla f(\mathbf{x})$, то такой алгоритм называется *алгоритмом первого порядка*, если дополнительно требуется умение вычислять ещё и матрицу Гессе вторых производных H, то такой алгоритм — *второго порядка*. С одной стороны, интуитивно ожидается, что чем больше информации задействовано в алгоритме, тем меньше потребуется итераций для достижения заданной точности. Либо, например, если известно, что целевая функция обладает седловыми точками, то потребуется использовать алгоритм, сходящийся к критической точке второго рода, который очевидно окажется алгоритмом второго порядка. С другой стороны, вовлечение матрицы вторых производных может быть практически затруднено при решении задач оптимизации очень большой размерности, например, при обучении нейронных сетей, где размерность вектора $\mathbf{x}$ может достигать десятков миллионов, а значит матрица вторых производных состояла бы из порядка 10^{14} элементов. Либо,

например, вычисление градиента или матрицы производных затруднено, например, по причине громоздких формул. В последнем случае могут использоваться приёмы, аппроксимирующие первые и вторые производные с использованием конечных разностей.

В-четвёртых, алгоритм может учитывать или не учитывать ограничения. Если требуется решить задачу условной оптимизации и известно, что потенциальное решение может лежать на границе допустимого множества, то необходимо применять алгоритм, который учитывает ограничения. При применении проекционных методов условной оптимизации (см. раздел 4.3.1) необходимо учитывать, что каждый проекционный метод работает только с допустимыми множествами определённой формы, например:

- положительный ортант $\mathbf{x} \geq \mathbf{0}$,
- координатный параллелепипед[7] $\mathbf{a} \leq \mathbf{x} \leq \mathbf{b}$,
- полиэдр $A\mathbf{x} \geq \mathbf{b}$,
- внутренность шара $\|\mathbf{x} - \mathbf{x}_0\| \leq R$.

Сравнения векторов, как и ранее в этой главе, производятся поэлементно и обозначают систему неравенств.

Скорость работы итеративного алгоритма оптимизации может быть теоретически охарактеризована с двух точек зрения. Во-первых, можно охарактеризовать скорость работы одного шага в привычных терминах O-нотации в зависимости от размерности пространства n в котором решается задача.

Во-вторых, можно охарактеризовать количество требуемых шагов для достижения результата. Обычно различают следующие классы алгоритмов оптимизации по скорости сходимости производимых ими последовательностей. Последовательность $\mathbf{x}^{(k)}$ сходится с *линейной скоростью* в точку $\mathbf{x}^*$, если

$$\exists q \in (0,1), k_0 : \quad \|\mathbf{x}^{(k+1)} - \mathbf{x}^*\| \leq q\|\mathbf{x}^{(k)} - \mathbf{x}^*\| \quad \forall k \geq k_0. \tag{4.29}$$

Последовательность $\mathbf{x}^{(k)}$ сходится с *сверхлинейной скоростью* в точку $\mathbf{x}^*$, если

$$\|\mathbf{x}^{(k+1)} - \mathbf{x}^*\| \leq q_k\|\mathbf{x}^{(k)} - \mathbf{x}^*\| \quad q_k \to 0 \quad k \to \infty. \tag{4.30}$$

Последовательность $\mathbf{x}^{(k)}$ сходится с *квадратичной скоростью* в точку $\mathbf{x}^*$, если

$$\exists q > 0, k_0 : \quad \|\mathbf{x}^{(k+1)} - \mathbf{x}^*\| \leq q\|\mathbf{x}^{(k)} - \mathbf{x}^*\|^2 \quad \forall k \geq k_0. \tag{4.31}$$

Такой подход удобен для случаев, когда формально алгоритму требуется бесконечное число шагов. Понятно, что на практике алгоритм оптимизации, формально сходящийся за бесконечное число шагов, принудительно завершается при выполнении того или иного критерия, например:

[7] box

- скорость убывания целевой функции $f(\mathbf{x}^{(k)}) - f(\mathbf{x}^{(k+1)})$ ниже некоторого порога,
- норма вектора изменения параметров $\|\mathbf{x}^{(k)} - \mathbf{x}^{(k+1)}\|$ ниже некоторого порога,
- норма градиента функции $\|\nabla f(\mathbf{x}^{(k+1)})\|$ ниже некоторого порога.

Сравнение в терминах скорости сходимости позволяет исключить из рассмотрения конкретный выбранный критерий остановки.

На практике, скорость работы алгоритма оптимизации это баланс между двумя крайними случаями: малым числом шагов, вычисление каждого из которых занимает значительное время, и большим числом шагов, вычисление каждого из которых не требует значительных затрат. Поэтому при необходимости решать большое число однотипных задач оптимизации, имеет смысл произвести замер скорости работы реализации алгоритма, выбранного для решения задачи. В качестве альтернативы измерению времени по секундомеру часто используется подсчёт числа точек, в которых алгоритму потребовалось вычислять значение целевой функции $f(\mathbf{x})$. Такой подход наиболее полезен в ситуациях, когда вычисление целевой функции $f(\mathbf{x})$ само по себе является операцией, требующей значительных вычислительных затрат: возможность вычислить функцию $f(\mathbf{x})$ не подразумевает, что она должна быть задана в виде формулы. Например, вычисление вектор-функции нелинейной модели $\phi(\mathbf{x})$ из нелинейной задачи наименьших квадратов (4.24) может потребовать численного решения системы дифференциальных уравнений, если идёт речь о модели движения некоторой системы, параметры $\mathbf{x}$ которой не известны, а координаты которой в разные моменты времени представлены вектором измерений $\hat{\mathbf{b}}$.

Итеративная природа алгоритмов подразумевает вопрос не только о точке куда сходится последовательность, но и о *начальном приближении* $\mathbf{x}^{(0)}$, откуда начинается процесс вычисления последовательности векторов $\mathbf{x}^{(k)}$. Кстати говоря, в ряде случаев удаётся формально доказать сходимость последовательности (т.е. сходимость алгоритма), только если начальное приближение находится в некоторой конечной окрестности предельной точки. Очевидно, что конкретное числовое значение предельной точки нам неизвестно, его то мы и ищем численно.

Когда начальное приближение $\mathbf{x}^{(0)}$ является параметром алгоритма, то на практике выбирается тем или иным подходящим способом. Если параметры $\mathbf{x}$ допускают физическую интерпретацию, то начальное приближение может быть выбрано, например, как оценка этих же параметров выполненная для более грубой модели, допускающей оценку параметров более простым способом, не требующим знание начального приближения. Например, линейная модель позволяет воспользоваться линейным методом наименьших квадратов (см. главу 2). Когда параметры $\mathbf{x}$ не допускают физической интерпретации, например, как в случае машинного обучения, то зачастую начальное приближение $\mathbf{x}^{(0)}$ выбирается случайным образом.

4.2.1 Алгоритмы безусловной оптимизации

Метод градиентного спуска

Методом градиентного спуска называется численный алгоритм оптимизации, в котором следующее приближение вычисляется по формуле

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \nabla f(\mathbf{x}^{(k)}), \tag{4.32}$$

где $\alpha_k > 0$ некоторый параметр, закон эволюции которого выбирается заранее. На практике часто выбирают $\alpha_k = \alpha_0$, или, например, $\alpha_k = \alpha_0/k$, где α_0 некоторая постоянная.

Идею метода градиентного спуска можно достаточно просто объяснить: вектор антиградиента $-\nabla f(\mathbf{x}^{(k)})$ показывает направление, в котором значение функции убывает. Это просто показать, разложив функцию $f(\mathbf{x})$ в ряд Тейлора в окрестности точки $\mathbf{x}^{(k)}$ следующим образом:

$$f(\mathbf{x}) = f(\mathbf{x}^{(k)}) + (\nabla f(\mathbf{x}^{(k)}), \mathbf{x} - \mathbf{x}^{(k)}) + o\left(\|\mathbf{x} - \mathbf{x}^{(k)}\|\right). \tag{4.33}$$

Для гладкой функции $f(\mathbf{x})$ можно выбрать шар $\|\mathbf{x}-\mathbf{x}^{(k)}\| \leq \rho$ настолько малым, что в этом шаре разложение (4.33) будет отличаться от функции $f(\mathbf{x})$ не более чем на наперёд заданную точность. Тогда минимум разложения (4.33) при условии $\|\mathbf{x} - \mathbf{x}^{(k)}\| \leq \rho$ можно выразить в следующем виде:

$$\mathbf{x}^* = \mathbf{x}^{(k)} - \frac{\rho}{\|\nabla f(\mathbf{x}^{(k)})\|} \nabla f(\mathbf{x}^{(k)}). \tag{4.34}$$

Значит, если предположить, что коэффициент α_k выбран достаточно малым, то можно ожидать, что

$$f(\mathbf{x}^{(k+1)}) = f\left(\mathbf{x}^{(k)} - \alpha_k \nabla f(\mathbf{x}^{(k)})\right) < f(\mathbf{x}^{(k)}). \tag{4.35}$$

Конечно, это только наши ожидания, на практике это утверждение выполняется далеко не всегда из-за того, что нельзя сказать заранее достаточно ли мал α_k. Однако, для каждого конкретного случая справедливость (4.35) можно проверить численно, и, если неравенство не выполняется, то уменьшить шаг $\alpha'_k = \beta\alpha_k$, где $0 < \beta < 1$. Такой подход часто называют методом дробления шага.

Удаётся доказать (см., например, Сухарев, Тимохов и Федоров 2008, или другие учебники по методам оптимизации), что если шаг α_k выбирается методом дробления так, что

$$f\left(\mathbf{x}^{(k)} - \alpha_k \nabla f(\mathbf{x}^{(k)})\right) \leq f(\mathbf{x}^{(k)}) - \epsilon\alpha_k \|\nabla f(\mathbf{x}^{(k)})\|^2, \tag{4.36}$$

где $\epsilon \in (0, 1)$, а градиент функции — Липшиц-непрерывен:

$$\|\nabla f(\mathbf{x}_1) - \nabla f(\mathbf{x}_2)\| \leq M\|\mathbf{x}_1 - \mathbf{x}_2\| \quad \forall \mathbf{x}_1, \mathbf{x}_2 \in \mathbf{R^n}, \tag{4.37}$$

тогда последовательность векторов $\mathbf{x}^{(k)}$, построенных по схеме 4.32, независимо от выбора начального приближения $\mathbf{x}^{(0)}$ сходится к критической точке первого рода безусловной задачи оптимизации (4.3).

Алгоритм Ньютона

Алгоритмом (методом) Ньютона называется численный метод оптимизации с шагом:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - H(\mathbf{x}^{(k)})^{-1}\nabla f(\mathbf{x}^{(k)}), \tag{4.38}$$

где $H(\mathbf{x}^{(k)})$ — матрица Гессе, вычисленная в точке $\mathbf{x}^{(k)}$. В отличии от метода градиентного спуска, вектор $\nabla f(\mathbf{x}^{(k)})$ умножается не на константу, а на обратную матрицу вторых производных.

Идея метода аналогична методу градиентного спуска, но вместо разложения (4.33) используется разложение до второго порядка малости:

$$f(\mathbf{x}) = f(\mathbf{x}^{(k)}) + (\nabla f(\mathbf{x}^{(k)}), \mathbf{x} - \mathbf{x}^{(k)}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(k)}, H(\mathbf{x} - \mathbf{x}^{(k)})) + o\left(\|\mathbf{x} - \mathbf{x}^{(k)}\|^2\right). \tag{4.39}$$

Разложение (4.39) есть квадратичная форма, минимум которой может быть найден по известной формуле аналогично главе 2:

$$\mathbf{x}^{*} = \mathbf{x}^{(k)} - H(\mathbf{x}^{(k)})^{-1}\nabla f(\mathbf{x}^{(k)}), \tag{4.40}$$

однако требуется, чтобы матрица вторых производных H была положительно определённой, иначе разложение (4.39) не ограничено снизу.

При ряде условий удаётся доказать сходимость метода к минимуму выпуклых функций с квадратичной скоростью (см., например, Сухарев, Тимохов и Федоров 2008). К недостаткам метода относят необходимость вычислять и обращать матрицу вторых производных на каждом шаге. Кроме того, $\mathbf{x}^{(k+1)}$, вычисленный по схеме (4.38), может оказаться настолько далеко от точки $\mathbf{x}^{(k)}$, что само разложение (4.39) в этой точке неадекватно аппроксимирует целевую функцию $f(\mathbf{x}^{(k+1)})$, а значит такой шаг может привести не к уменьшению, а к росту целевой функции за счёт неправильной аппроксимации. Одной из модификаций алгоритма является алгоритм Ньютона с регулировкой шага:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k H(\mathbf{x}^{(k)})^{-1}\nabla f(\mathbf{x}^{(k)}), \tag{4.41}$$

где α_k — параметр, имеющий смысл, аналогичный одноимённому параметру в методе градиентного спуска.

Алгоритм Гаусса–Ньютона

Алгоритм Гаусса–Ньютона — это адаптация алгоритма Ньютона для решения задачи нелинейных наименьших квадратов (4.24):

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - (J^T J)^{-1} J^T \left(\phi(\mathbf{x}^{(k)}) - \hat{\mathbf{b}}\right). \tag{4.42}$$

В этом уравнении

$$\nabla\left(\frac{1}{2}\|\phi(\mathbf{x}^{(k)}) - \hat{\mathbf{b}}\|^2\right) = J^T\left(\phi(\mathbf{x}^{(k)}) - \hat{\mathbf{b}}\right), \tag{4.43}$$

где J обозначает матрицу Якоби вектор-функции $\phi(\mathbf{x}^{(k)})$, вычисленную в точке $\mathbf{x}^{(k)}$. Вычисления вторых производных функции $\phi(\mathbf{x}^{(k)})$ в этом методе не требуется, так как вместо матрицы H стоит произведение $J^T J$, чья симметричная форма гарантирует неотрицательную определённость матрицы.

Метод основан на разложении в ряд Тейлора вектор-функции $\phi(\mathbf{x})$ в окрестности точки $\mathbf{x}^{(k)}$:

$$\phi(\mathbf{x}) = \phi(\mathbf{x}^{(k)}) + J(\mathbf{x} - \mathbf{x}^{(k)}) + o\left(\|\mathbf{x} - \mathbf{x}^{(k)}\|\right). \tag{4.44}$$

Подставляя разложение (4.44) в целевую функцию задачи нелинейных наименьших квадратов (4.24) получим следующую аппроксимацию целевой функции $f(\mathbf{x})$:

$$f_k(\mathbf{x}) = \frac{1}{2}\|\phi(\mathbf{x}^{(k)}) - \mathbf{b}\|^2 + (J^T(\phi(\mathbf{x}^{(k)}) - \mathbf{b}), \mathbf{x} - \mathbf{x}^{(k)}) + \frac{1}{2}(\mathbf{x} - \mathbf{x}^{(k)}, J^T J(\mathbf{x} - \mathbf{x}^{(k)})). \tag{4.45}$$

Аппроксимация (4.45) не является разложением в ряд Тейлора, так как нужно либо проигнорировать третье слагаемое, которое имеет второй порядок малости по отношению к разложению (4.44), либо добавить ещё одно слагаемое такого же порядка малости, включающее вторые производные вектор-функции $\phi(\mathbf{x})$. Тем не менее, аналогично рассуждениям метода Ньютона находится минимум этой квадратичной формы, что и приводит к выражению (4.42).

Алгоритм Левенберга–Марквардта

Общей проблемой методов градиентного спуска (4.32), Ньютона (4.38) и Гаусса–Ньютона (4.42) является потенциально большие смещения, которые могут возникнуть на одном или нескольких шагах. Большим смещением $\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}$ будем считать такое смещение, при котором

$$f(\mathbf{x}^{(k+1)}) > f(\mathbf{x}^{(k)}), \tag{4.46}$$

что является следствием неадекватности используемой аппроксимации целевой функции на данном смещении относительно точки разложения $\mathbf{x}^{(k)}$. Понятно, что аппроксимации (4.33), (4.39) и (4.44) идеально точно совпадают с целевой функцией $f(\mathbf{x})$ только в точке разложения $\mathbf{x}^{(k)}$, и, чем дальше мы отступаем от точки разложения, тем больше шансов столкнуться с ситуацией (4.46).

Американские исследователи Кеннет Левенберг (Levenberg 1944) и Дональд Марквардт (Marquardt 1963) независимо предложили способ разрешения этой проблемы, альтернативный подходу дробления шага или методу Ньютона с регулировкой шага (4.41). Предложенный ими метод решения задачи нелинейных наименьших квадратов (4.24) впоследствии был назван алгоритмом Левенберга–Марквардта.

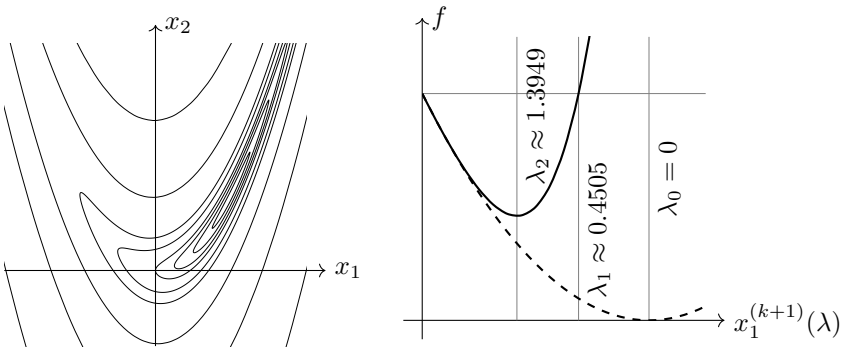

Рис. 4.2: Применение метода Левенберга-Марквардта для поиска минимума функции $f(\mathbf{x}) = (1 - x_1)^2 + 4(x_2 - x_1^2)^2$. Слева показан контурный график целевой функции. Справа показан пример поведения целевой функции и её квадратичной аппроксимации методом Левенберга-Марквардта в точке $\mathbf{x}^{(k+1)}(\lambda)$ для $\mathbf{x}^{(k)} = \mathbf{0}$ и различных значений параметра λ. Сплошная линия показывает поведение целевой функции вдоль $\mathbf{x}^{(k+1)}(\lambda)$, штриховая линия показывает поведение квадратичной аппроксимации целевой функции. Случай $\lambda_0 = 0$ соответствует применению шага Гаусса–Ньютона (4.42).

Интересно, что авторы рассуждали по разному, но пришли в конечном итоге к одному и тому же методу. Левенберг предложил следующую модификацию схемы метода Гаусса–Ньютона (4.42):

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \arg\min_{\boldsymbol{\Delta x}\in\mathbf{R^n}} \left(\left\| J\boldsymbol{\Delta x} - \left(\phi(\mathbf{x}^{(k)}) - \mathbf{b}\right) \right\|^2 + \frac{1}{w}\|\boldsymbol{\Delta x}\|^2 \right), \quad (4.47)$$

или в явном виде для сравнения с (4.42):

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \left(J^T J + \frac{1}{w} I \right)^{-1} J^T \left(\phi(\mathbf{x}^{(k)}) - \hat{\mathbf{b}} \right), \quad (4.48)$$

где I — единичная матрица, якобиан J вычисляется в точке $\mathbf{x}^{(k)}$, а параметр w подбирается таким образом, чтобы $f(\mathbf{x}^{(k+1)}(w)) < f(\mathbf{x}^{(k)})$. Самый простой способ подбора w состоит в последовательном вычислении кандидатов (4.48) на некоторой сетке. Заметим, что эти рассуждения перекликаются с подходом регуляризации Тихонова для линейного метода наименьших квадратов, фактически выражения (4.47) и (4.48) с точностью до обозначений совпадают с соответствующими выражениями (2.42) и (2.43).

Марквардт предложил выбирать наилучшее смещение $\boldsymbol{\Delta x}$ среди всех векторов с фиксированной нормой $\|\boldsymbol{\Delta x}\| = \Delta$, т.е. на каждом решать искать

минимум квадратичной формы на сфере радиуса Δ:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} + \mathbf{\Delta x}^*, \tag{4.49}$$

$$\mathbf{\Delta x}^* = \arg \min_{\mathbf{\Delta x} \in \mathbf{R}^n} \left\| J\mathbf{\Delta x} - \left(\phi(\mathbf{x}^{(k)}) - \mathbf{b}\right)\right\|^2, \tag{4.50}$$

$$\|\mathbf{\Delta x}\|^2 = \Delta^2, \tag{4.51}$$

что в обозначениях множителей Лагранжа приводит к такой же схеме метода, как и (4.48):

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \left(J^T J + \lambda I\right)^{-1} J^T \left(\phi(\mathbf{x}^{(k)}) - \hat{\mathbf{b}}\right). \tag{4.52}$$

Неявный параметр радиуса сферы Δ монотонно убывает с ростом множителя Лагранжа λ, который предлагается подбирать на каждом шаге согласно следующей процедуре. Пусть $\nu > 1$ — некоторый постоянный параметр, а $\lambda^{(k)}$ обозначает величину множителя Лагранжа λ, которая была выбрана на предыдущем шаге. Начальное значение $\lambda^{(0)} > 0$ может быть выбрано произвольно, обычно берётся значение меньше единицы. Тогда рассчитаем два пробных кандидата по формуле (4.52) для двух разных значений λ: для шага предыдущего размера $\lambda^{(k)}$ и для увеличенного шага $\lambda^{(k)}/\nu$, а затем сравним значение целевой функции в точках $\mathbf{x}^{(k+1)}(\lambda^{(k)})$, $\mathbf{x}^{(k+1)}(\lambda^{(k)}/\nu)$ и $\mathbf{x}^{(k)}$:

- если
$$f(\mathbf{x}^{(k+1)}(\lambda^{(k)}/\nu)) \le f(\mathbf{x}^{(k)}),$$
значит бóльший шаг улучшил приближение, поэтому выбирается более длинный шаг: $\lambda^{(k+1)} = \lambda^{(k)}/\nu$, $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k+1)}(\lambda^{(k)}/\nu)$;

- если
$$f(\mathbf{x}^{(k+1)}(\lambda^{(k)})) \le f(\mathbf{x}^{(k)}),$$
$$f(\mathbf{x}^{(k+1)}(\lambda^{(k)}/\nu)) > f(\mathbf{x}^{(k)}),$$
значит шаг текущего размера улучшает приближение, а бóльший шаг ухудшает приближение, поэтому выбирается шаг с консервативной длиной: $\lambda^{(k+1)} = \lambda^{(k)}$, $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k+1)}(\lambda^{(k)})$;

- если
$$f(\mathbf{x}^{(k+1)}(\lambda^{(k)}/\nu)) > f(\mathbf{x}^{(k)}),$$
$$f(\mathbf{x}^{(k+1)}(\lambda^{(k)})) > f(\mathbf{x}^{(k)}),$$
значит что оба шага слишком большие, поэтому необходимо выбрать уменьшенный шаг: $\lambda^{(k+1)} = \lambda^{(k)}\nu^w$, $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k+1)}(\lambda^{(k)}\nu^w)$, где w некоторая целая степень, такая, что $f(\mathbf{x}^{(k+1)}(\lambda^{(k)}\nu^w)) \le f(\mathbf{x}^{(k)})$.

Пример поведения $f(\mathbf{x}^{(k+1)}(\lambda))$ приводится на Рис. 4.2, где видно, что слишком маленькие значения λ соответствуют области, в которой квадратичная аппроксимация описывает целевую функцию неадекватно, а слишком большие значения λ, напротив, приводят к маленьким шагам и медленному продвижению алгоритма.

Предложенный способ одновременного выбора $\mathbf{x}^{(k+1)}$ и $\lambda^{(k+1)}$ позволяет динамически изменять размер шага в зависимости от локальных особенностей целевой функции $f(\mathbf{x})$. Иными словами, поддерживается оптимальный размер шага: с одной стороны он достаточно мал, чтобы аппроксимация целевой функции квадратичной формой работала корректно на каждом этапе, с другой стороны шаг достаточно велик, чтобы увеличивать скорость сходимости за счёт уменьшения числа шагов.

Методы доверительных областей

Существует целый класс методов, объединённых под общим названием *методы доверительных областей*[8], пожалуй, наиболее полное описание которых дано в монографии Conn, Gould и Toint 2000. Можно усмотреть аналогии между этих подходом и методом Левенберга–Марквардa, хотя методы доверительных областей гораздо богаче и позволяют решать не только нелинейную задачу наименьших квадратов (4.24), но и общую задачу безусловной оптимизации (4.3), задачи условной оптимизации (4.4), и т.д. Во-первых, как и раньше целевая функция $f(\mathbf{x})$ в каждой точке $\mathbf{x}^{(k)}$ заменяется аппроксимацией $f_k(\mathbf{x})$, которая, например, в зависимости от решаемой задачи и выбранного метода может быть линейной (4.33), квадратичной вида (4.39) или квадратичной вида (4.44), если решается нелинейная задача наименьших квадратов, или любой другой удобной аппроксимацией.

Во-вторых, считается, что выбранная аппроксимация действительна (адекватна) лишь в окрестности точки разложения, которую называют *доверительной областью*, и размер Δ_k которой явно отслеживается и динамически изменяется. Например, доверительная область чаще всего задаётся в виде многомерного шара: $\|\mathbf{x} - \mathbf{x}^{(k)}\| \leq \Delta_k$, но может быть задана и в виде координатного параллелепипеда $\|\mathbf{x} - \mathbf{x}^{(k)}\|_\infty \leq \Delta_k$. Следующее приближение $\mathbf{x}^{(k+1)}$ может быть выбрано любым способом, но только внутри доверительной области.

Например, линейная и квадратичные аппроксимации допускают решение следующих условных задач оптимизации:

$$\mathbf{x}^{(k+1)} = \arg\min_{\mathbf{x}\in\mathbf{R^n}} f_k(\mathbf{x}), \tag{4.53}$$

$$\|\mathbf{x} - \mathbf{x}^{(k)}\|^2 \leq \Delta_k^2. \tag{4.54}$$

В случае линейной аппроксимации $\mathbf{x}^{(k+1)}$ находится по формуле (4.34).

В случае квадратичной аппроксимации задача сводится к поиску собственных значений матрицы H аппроксимирующей квадратичной формы

[8] Trust-region method

и численному решению нелинейного уравнения относительно множителя Лагранжа λ, а затем $\mathbf{x}^{(k+1)}$ вычисляется по знакомой формуле:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - (H + \lambda I)^{-1} \nabla f(\mathbf{x}^{(k)}), \tag{4.55}$$

где I — единичная матрица, а множитель Лагранжа $\lambda \geq 0$ найден ранее. Напомним, что если безусловный минимум квадратичной формы лежит внутри доверительной области, то множитель Лагранжа $\lambda = 0$.

Если решается нелинейная задача наименьших квадратов, то

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \left(J^T J + \lambda I\right)^{-1} J^T \left(\phi(\mathbf{x}^{(k)}) - \hat{\mathbf{b}}\right), \tag{4.56}$$

но в отличии от метода Левенберга–Марквардта (4.52), $\lambda \geq 0$ вычисляется заранее исходя из условия $\|\mathbf{x}^{(k+1)} - \mathbf{x}^{(k)}\|^2 \leq \Delta_k^2$. Такие вычисления сопряжены с необходимостью отыскания собственных значений матрицы H. Кроме того, благодаря ограничению (4.54) матрица H аппроксимирующей квадратичной формы может быть отрицательно определена, чего нельзя было допустить в методе Ньютона. Квадратичная форма с отрицательно определённой матрицей H имеет условный минимум на поверхности сферы $\|\mathbf{x} - \mathbf{x}^{(k)}\| = \Delta_k$, и этот минимум может быть найден с помощью собственных векторов матрицы H.

Более того, вспоминая, что мы имеем дело с аппроксимацией $f_k(\mathbf{x})$ вместо истинной целевой функции $f(\mathbf{x})$, можно предложить способ вычисления $\mathbf{x}^{(k+1)}$ на основе приближенного решения задачи (4.53)-(4.54). Один из эффективных способов приближенного поиска $\mathbf{x}^{(k+1)}$ основан на методе сопряжённых градиентов (см. раздел 4.2.2).

В-третьих, размер доверительной области Δ_k на каждом шаге контролируется на основе количественной меры:

$$\rho_k = \frac{f(\mathbf{x}^{(k+1)}) - f(\mathbf{x}^{(k)})}{f_k(\mathbf{x}^{(k+1)}) - f_k(\mathbf{x}^{(k)})}, \tag{4.57}$$

где в числителе стоит разность значений целевой функции, а в знаменателе — разность значений аппроксимации. В идеальном случае значение ρ_k должно быть близко к 1, однако в общем случае ρ_k может быть как положительной так и отрицательной величиной. Количественная мера ρ_k используется для адаптации размера доверительной области Δ_k, например следующим образом:

- если $\rho_k \in [1 - \alpha_1, 1 + \alpha_1]$, где α_1 некоторый постоянный порог, значит аппроксимация работает хорошо, и чтобы ускорить сходимость, для следующей итерации Δ_{k+1} выбирается больше, чем Δ_k;

- если $\rho_k \in [1 - \alpha_2, 1 + \alpha_2]$, где $\alpha_2 > \alpha_1$ некоторый постоянный порог, значит аппроксимация работает удовлетворительно, и размер доверительной области сохраняется $\Delta_{k+1} = \Delta_k$;

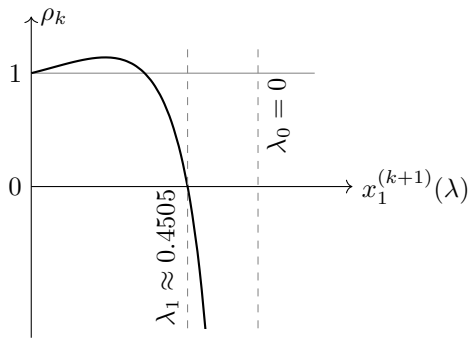

Рис. 4.3: Поведение функции ρ_k для случая Рис. 4.2: рассматривается квадратичная аппроксимация функции $f(\mathbf{x}) = (1 - x_1)^2 + 4(x_2 - x_1^2)^2$ в точке $\mathbf{x}^{(k)} = \mathbf{0}$.

- если $|\rho_k - 1| > \alpha_2$, значит был выбран слишком большой размер доверительной области, и для следующего шага Δ_{k+1} выбирается меньше, чем Δ_k, причём, если $\rho_k < 0$, то дополнительно $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$.

На Рис. 4.3 приводится пример поведения функции ρ_k.

4.2.2 Метод сопряжённых градиентов

Вернёмся к задаче квадратичного программирования (4.20) и зададимся следующим вопросом. Пусть у нас есть линейное подпространство, в котором столбцы матрицы $S_k \in \mathbf{R}^{\mathbf{n}\times\mathbf{k}}$ задают базис, такой, что $S_k^T H S_k$ невырожденная матрица. Рассмотрим аффинное преобразование $\mathbf{x} = \mathbf{x}^{(0)} + S_k\mathbf{s}$, где $\mathbf{s} \in \mathbf{R}^{\mathbf{k}}$, и найдём точку минимума квадратичной формы среди всех $\mathbf{x}$ удовлетворяющих этому условию. Можно доказать[9], что искомая точка условного минимума выражается формулой:

$$\mathbf{x}^* = \mathbf{x}^{(0)} - S_k \left(S_k^T H S_k\right)^{-1} S_k^T \nabla f(\mathbf{x}^{(0)}), \tag{4.58}$$

и удовлетворяет следующему условию

$$S_k^T \nabla f(\mathbf{x}^*) = 0, \tag{4.59}$$

которое можно прочитать, как условие ортогональности вектора градиента $\nabla f(\mathbf{x}^*)$ выбранному линейному подпространству. Условие (4.59) полностью соответствует геометрической интерпретации условий Каруша-Куна-Такера (4.14).

Теперь рассмотрим вложенную последовательность линейных подпространств возрастающей размерности, т.е. базис каждого нового подпространства S_{k+1} получается с помощью добавления нового базисного вектора $\mathbf{s}^{(k)}$ к существующему базису S_k. Тогда соотношение (4.58) с учётом

[9] Достаточно рассмотреть $\nabla_s f(\mathbf{x}^{(0)} + S\mathbf{s}) = 0$, а затем заменить все вхождения $\mathbf{c} + H\mathbf{x}^{(0)}$ назад на $\nabla f(\mathbf{x}^{(0)})$.

свойства (4.59) перепишется в виде:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - S_{k+1}\left(S_{k+1}^T H S_{k+1}\right)^{-1} S_{k+1}^T \nabla f(\mathbf{x}^{(k)}) \\ = \mathbf{x}^{(k)} - (\mathbf{s}^{(k)}, \nabla f(\mathbf{x}^{(k)}))S_{k+1}\left(S_{k+1}^T H S_{k+1}\right)^{-1}\mathbf{e}_k, \quad (4.60)$$

где $\mathbf{x}^{(k+1)}$ — точка минимума для пространства размерности $k+1$, $\mathbf{x}^{(k)}$ — точка минимума для пространства размерности k, $\mathbf{s}^{(k)}$ новый дополнительный вектор базиса, а $\mathbf{e}_k$ обозначает орт с ненулевым последним компонентом.

Пусть S_k состоит из H-*сопряжённых*[10] колонок. H-сопряжёнными, или просто сопряжёнными, будем называть такие вектора $\mathbf{p}^{(i)}$, для которых $(\mathbf{p}^{(i)}, H\mathbf{p}^{(j)}) = 0$ для $\forall i, j : i \neq j$. Тогда соотношение (4.60) перепишется в следующем виде:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \frac{(\mathbf{p}^{(k)}, \nabla f(\mathbf{x}^{(k)}))}{(\mathbf{p}^{(k)}, H\mathbf{p}^{(k)})}\mathbf{p}^{(k)}, \quad 0 \leq k < n, \quad (4.61)$$

а выражение для градиента

$$\nabla f(\mathbf{x}^{(k+1)}) = \nabla f(\mathbf{x}^{(k)}) - \frac{(\mathbf{p}^{(k)}, \nabla f(\mathbf{x}^{(k)}))}{(\mathbf{p}^{(k)}, H\mathbf{p}^{(k)})}H\mathbf{p}^{(k)}, \quad 0 \leq k < n. \quad (4.62)$$

Такая рекуррентная схема обладает целым рядом полезных свойств. Во-первых, схема за n шагов сходится к точке безусловного минимума квадратичной формы. Заметим, что алгоритм устойчив в том смысле, что при продолжении расчёта для $k \geq n$ получаемая последовательность $\mathbf{x}^{(k)}$ сходится к точке безусловного минимума квадратичной формы. Расчёт для $k \geq n$ используют для устранения ошибок округления, накопленных по мере расчёта.

Во-вторых, схема не требует нахождения обратной матрицы H^{-1}, алгоритм предполагает умение умножать матрицу H на любой вектор, что позволяет оптимизировать хранение матрицы H в памяти, если структура матрицы имеет какую-то особую форму.

В-третьих, на каждом шаге алгоритма значение целевой функции монотонно убывает, значит алгоритм можно оборвать на любом шаге, когда задача квадратичного программирования используется в составе итеративного алгоритма численной оптимизации.

В-четвёртых, на каждом шаге итеративного алгоритма требуется дополнительно хранение только нескольких векторов размера n, т.е. требование по дополнительной памяти $O(n)$.

Однако, пока остаётся непонятным как эффективно построить H-сопряжённый вложенный базис $\mathbf{p}^{(k)}$. К счастью, H-сопряжённый базис можно легко построить прямо по ходу расчёта, например, по следующей рекуррентной схеме:

$$\mathbf{p}^{(k+1)} = -\nabla f(\mathbf{x}^{(k+1)}) + \frac{(\nabla f(\mathbf{x}^{(k+1)}), H\mathbf{p}^{(k)})}{(\mathbf{p}^{(k)}, H\mathbf{p}^{(k)})}\mathbf{p}^{(k)}, \quad 0 \leq k < n-1, \quad (4.63)$$

[10] *conjugated*

а $\mathbf{p}^{(0)} = -\nabla f(\mathbf{x}^{(0)})$, где $\mathbf{x}^{(0)}$ некоторая начальная точка, например, $\mathbf{x}^{(0)} = 0$, тогда $\mathbf{p}^{(0)} = -\mathbf{c}$.

Формула (4.63) нуждается в пояснении. Предположим, что существует $n' \leq n$ такое, что

$$(\mathbf{p}^{(i)}, H\mathbf{p}^{(j)}) = 0 \quad \forall i, j : 0 \leq i, j < n', i \neq j, \tag{4.64}$$

т.е. формулы (4.61), (4.62) и (4.63) справедливы для $n = n'$, тогда выполняется следующее соотношение:

$$(\nabla f(\mathbf{x}^{(k+1)}), \nabla f(\mathbf{x}^{(j)})) = 0 \quad \forall j, k : 0 \leq j \leq k < n', \tag{4.65}$$

иначе говоря, градиенты квадратичной формы, взятые в первых n' точках взаимоортогональны.

- Для $j = 0$ это утверждение следует из $\nabla f(\mathbf{x}^{(0)}) = -\mathbf{p}^{(0)}$ и формулы (4.59).
- Для $j > 0$ следует скалярно домножить формулу (4.63), записанную для $k+1 = j$, на $\nabla f(\mathbf{x}^{(k+1)})$: тогда слагаемое слева и второе слагаемое справа окажутся равными нулю в силу формулы (4.59), следовательно, равно нулю и оставшееся слагаемое, равное $(\nabla f(\mathbf{x}^{(k+1)}), \nabla f(\mathbf{x}^{(j)}))$, где $1 \leq j < n'$.

Теперь вспомним, что вектор $-\nabla f(\mathbf{x}^{(n')})$ и вектора $\mathbf{p}^{(k)}$ линейно независимы, значит можно построить новый H-сопряжённый базисный вектор $\mathbf{p}^{(n')}$ с помощью процедуры ортоганализации Грама-Шмидта:

$$\mathbf{p}^{(n')} = -\nabla f(\mathbf{x}^{(n')}) - \sum_{k=0}^{n'-1} \frac{(\nabla f(\mathbf{x}^{(n')}), H\mathbf{p}^{(k)})}{(\mathbf{p}^{(k)}, H\mathbf{p}^{(k)})} \mathbf{p}^{(k)}. \tag{4.66}$$

Заметим, однако, что $H\mathbf{p}^{(k)}$ можно выразить из формулы (4.62), затем воспользоваться свойством (4.65) и показать, что в сумме правой части (4.66) ненулевым является только слагаемое с $k = n' - 1$. Значит, $\mathbf{p}^{(n')}$ вычисленный по формуле (4.63) — H-сопряжённый ко всем остальным базисным векторам $\mathbf{p}^{(k)}$.

В случае $n' = 2$, скалярно домножая (4.63) при $k = 0$ на $H\mathbf{p}^{(0)}$, можно доказать, что $(\mathbf{p}^{(1)}, H\mathbf{p}^{(0)}) = 0$, откуда по индукции следует, что рассчитанный по формулам (4.61), (4.62) и (4.63) базис $\mathbf{p}^{(k)}$ — H-сопряжённый для всего рассматриваемого линейного пространства размерности n.

Из формул (4.62) и (4.63) следует, что некоторый базис (не H-сопряжённый и не ортогональный) исследуемых вложенных линейных подпространств может быть представлен набором векторов:

$$\mathbf{p}^{(0)}, H\mathbf{p}^{(0)}, \ldots, H^{k-1}\mathbf{p}^{(0)}, \tag{4.67}$$

т.е. фактически базис задаётся начальным вектором $\mathbf{p}^{(0)}$ и квадратной матрицей H, которая последовательно возводится в разную степень. Постро-

енные по такому принципу линейные подпространства называются *подпространствами Крылова*[11], в честь академика А.Н. Крылова, русского математика и корабельного инженера.

Использование формул (4.62) и (4.63) — не единственный способ построить удобный базис в подпространствах Крылова. Известно несколько методов, объединённых названием методов подпространств Крылова. Например, Корнелий Ланцош, математик венгерского происхождения, нашёл изящный итеративный метод пригодный для решения задачи (4.53)–(4.54) с $f_k(\mathbf{x})$ в виде квадратичной формы (Lanczos 1950), причём матрица H не обязательно должна быть положительно определённой.

Метод Ланцоша во многом аналогичен методу сопряжённых градиентов и обладает всеми преимуществами последнего: требуется только операция умножения матрицы H на произвольный вектор, требуется дополнительно только $O(n)$ памяти, а вычисления сходятся за n итераций. Рекуррентные соотношения для построения базиса включают в себя три последовательных базисных вектора, а не два как в (4.63). Предложенный алгоритм позволяет отыскивать собственные вектора, отвечающие нескольким минимальным или максимальным собственным значениями матрицы H.

4.3 Алгоритмы решения ограниченных задач оптимизации

4.3.1 Проекционные методы

Семейство проекционных методов численного решения ограниченных задач оптимизации применяется к выпуклым целевым множествам X, не требует вычисления целевой функции $f(\mathbf{x})$ и её градиента $\nabla f(\mathbf{x})$ в точках $\mathbf{x} \notin X$, и основано на следующей геометрической интерпретации условий критической точки первого порядка. Если множество X выпукло и выполняется условие:

$$(\nabla f(\mathbf{x}^*), \mathbf{x} - \mathbf{x}^*) \geq 0 \quad \forall \mathbf{x} \in X, \tag{4.68}$$

тогда $\mathbf{x}^*$ критическая точка первого порядка. Следовательно, необходимое условия оптимальности первого рода можно переформулировать следующим образом: если $\mathbf{x}^*$ — локальное решение ограниченной задачи оптимизации, и X — выпуклое множество, тогда выполняется (4.68). Предположим, что $\mathbf{x}^*$ — локальное решение, но условие (4.68) не выполняется, т.е. $\exists \mathbf{x}_0$ такой, что $(\nabla f(\mathbf{x}^*), \mathbf{x}_0 - \mathbf{x}^*) < 0$. Тогда, во-первых, в силу выпуклости множества X весь отрезок $\mathbf{x}_\lambda = \lambda \mathbf{x}_0 + (1 - \lambda)\mathbf{x}^* \in X$, где $\lambda \in [0, 1]$. Во-вторых, подставляя $\mathbf{x}_\lambda$ в (4.68), нетрудно убедиться, что условие не выполняется для всего отрезка, кроме самой точки $\mathbf{x}^*$. Раскладывая целевую функцию в ряд Тейлора вдоль этого отрезка в окрестности $\lambda = 0$, получаем:

$$.f(\lambda \mathbf{x}_0 + (1 - \lambda)\mathbf{x}^*) - f(\mathbf{x}^*) = \lambda(\nabla f(\mathbf{x}^*), \mathbf{x}_0 - \mathbf{x}^*) + o(\lambda), \tag{4.69}$$

[11] Krylov subspace

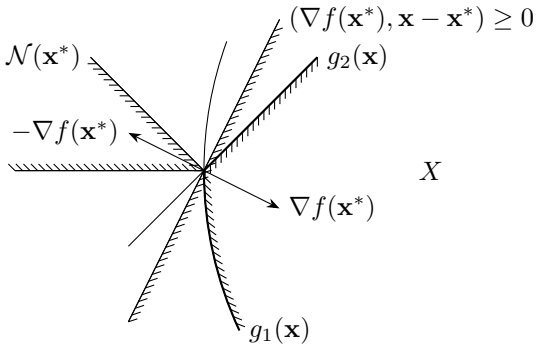

Рис. 4.4: Пример множества, заданного условием (4.68) в точке $\mathbf{x}^* = \mathbf{0}$, для задачи минимизации функции $f(\mathbf{x}) = (4x_1 - 2x_2 + 10)^2 + (x_1 + 2x_2)^2$ с ограничениями $g_1(\mathbf{x}) = -(x_1 - 1)^2 - x_2^2 + 1 \geq 0$, $g_2(\mathbf{x}) = x_1 - x_2 \geq 0$. Дополнительно показан *нормальный конус* $\mathcal{N}(\mathbf{x}^*)$, построенный на векторах $-\nabla g_1(\mathbf{x}^*)$ и $-\nabla g_2(\mathbf{x}^*)$. В соответствии с условием Каруша-Куна-Такера (4.10), вектор $-\nabla f(\mathbf{x}^*)$ лежит внутри нормального конуса.

откуда видно, что всегда можно подобрать такое достаточно малое λ_0, что $f(\lambda \mathbf{x}_0 + (1 - \lambda)\mathbf{x}^*) < f(\mathbf{x}^*)$ для $\lambda \in (0, \lambda_0)$, что в свою очередь противоречит исходному предположению о том, что $\mathbf{x}^*$ — локальное решение ограниченной задачи оптимизации. Геометрический смысл (4.68) поясняется на Рис. 4.4.

Условие (4.68) выглядит не слишком практичным с точки зрения реальных алгоритмов, поэтому рассмотрим вспомогательную задачу. Проекцией точки $\mathbf{a} \in \mathrm{R}^n$ на множество X называется точка $\pi_X(\mathbf{a})$, ближайшая к $\mathbf{a}$ из всех точек X:

$$\pi_X(\mathbf{a}) \equiv \arg\min_{\mathbf{x} \in X} \|\mathbf{x} - \mathbf{a}\|^2. \tag{4.70}$$

Для ряда простых множеств X можно предложить аналитическое выражение для $\pi_X(\mathbf{a})$. Далее нас будут интересовать только такие множества, для которых проекция вычисляется просто.

Например, если X — шар радиус R с центром в $\mathbf{x}_0$, тогда

$$\pi_X(\mathbf{a}) = \begin{cases} \mathbf{a} & \mathbf{a} \in X \\ \mathbf{x_0} + \frac{\mathbf{a} - \mathbf{x}_0}{\|\mathbf{a} - \mathbf{x}_0\|} R & \mathbf{a} \notin X. \end{cases} \tag{4.71}$$

Если X — неотрицательный ортант (т.е. $x_i \geq 0 \quad \forall i$), тогда

$$(\pi_X(\mathbf{a}))_i = \max\{0, a_i\}. \tag{4.72}$$

Более общий случай — координатный параллелепипед ($l_i \leq x_i \leq u_i \quad \forall i$, причём некоторые из l_i могут принимать значение $-\infty$, а некоторые из u_i могут принимать значение $+\infty$), проекция на который вычисляется следующим образом:

$$(\pi_X(\mathbf{a}))_i = \min\{\max\{l_i, a_i\}, u_i\} = \max\{l_i, \min\{a_i, u_i\}\}. \tag{4.73}$$

Пусть X — аффинное множество ($A\mathbf{x} = \mathbf{b} \quad \forall \mathbf{x} \in X$), тогда проекция запишется следующим образом:

$$\pi_X(\mathbf{a}) = \mathbf{a} - A^T(AA^T)^{-1}(A\mathbf{a} - \mathbf{b}). \tag{4.74}$$

Кроме того, для разнообразия отметим полезный случай, когда вместо аналитического выражения для проекции можно предложить простой алгоритм. Так, например, если искомый вектор $\mathbf{x}$ представляет некоторую монотонную зависимость, заданную на некоторой дискретной сетке (т.е. $x_1 \leq x_2 \leq ... \leq x_n$), тогда проекцию $\pi_X(\mathbf{a})$ можно вычислить за $O(n)$ операций (см. Best и Chakravarti 1990).

Проекция $\pi_X(\mathbf{a})$ обладает рядом следующих общих свойств, при условии, что X — выпуклое. Во-первых, проекция $\pi_X(\mathbf{a})$ существует для всех $\mathbf{a}$ и единственна, это следствие выпуклости самой функции $\|\mathbf{x} - \mathbf{a}\|^2$. Во-вторых, выполняется условие

$$(\pi_X(\mathbf{a}) - \mathbf{a}, \mathbf{x} - \pi_X(\mathbf{a})) \geq 0 \quad \forall \mathbf{x} \in X, \tag{4.75}$$

что является следствием (4.68). В-третьих, при проецировании, расстояние между любыми двумя точками $\mathbf{a}_1$ и $\mathbf{a}_2$ не возрастает:

$$\|\pi_X(\mathbf{a}_1) - \pi_X(\mathbf{a}_2)\| \leq \|\mathbf{a}_1 - \mathbf{a}_2\|, \tag{4.76}$$

что является следствием утверждения (4.75) применённого к $\pi_X(\mathbf{a}_1)$ и $\pi_X(\mathbf{a}_2)$, и неравенства Коши-Буняковского.

Теперь можно сформулировать окончательное условие критической точки первого порядка для выпуклого допустимого множества X в терминах проекции:

$$\mathbf{x}^* = \pi_X\left(\mathbf{x}^* - \lambda \nabla f(\mathbf{x}^*)\right) \quad \forall \lambda > 0, \tag{4.77}$$

где точка $\mathbf{x}^*$ — критическая точка первого порядка. Действительно, с одной стороны, при рассмотрении точки $\mathbf{a} = \mathbf{x}^* - \lambda \nabla f(\mathbf{x}^*)$ из (4.75) и (4.77) следует геометрическое определения критической точки (4.68).

С другой стороны, рассмотрим вектор $\pi_X(\mathbf{a}) - \mathbf{x}^*$, тогда, во-первых, из (4.68) следует $(\nabla f(\mathbf{x}^*), \pi_X(\mathbf{a}) - \mathbf{x}^*) \geq 0$, и, во-вторых, из (4.75) следует $\|\pi_X(\mathbf{a}) - \mathbf{x}^*\|^2 \leq -\lambda(\nabla f(\mathbf{x}^*), \pi_X(\mathbf{a}) - \mathbf{x}^*)$, следовательно, выполняется (4.77).

Метод проекции градиента

Условие (4.77) — основа проекционных методов численного решения ограниченных задач оптимизации. Например, можно предложить метод проекции градиента, адаптировав метод градиентного спуска (4.32) следующим образом:

$$\mathbf{x}^{(k+1)} = \pi_X\left(\mathbf{x}^{(k)} - \alpha_k \nabla f(\mathbf{x}^{(k)})\right). \tag{4.78}$$

Сходимость такого метода объясняется тем, что при определённых условиях (4.78) является сжимающим отображением, для которого должна существовать неподвижная точка $\mathbf{x}^*$, удовлетворяющая условию (4.77).

Метод наименьших квадратов с ограничениями неотрицательности параметров

В качестве другого примера приведём метод, предложенный для численного решения неотрицательной задачи наименьших квадратов[12] (4.22)-(4.23), см. Лоусон и Хенсон 1986. Заметим, что ограничения вида (4.23) разбивают R^n на 2^n подмножеств. Мы могли бы в каждом из таких подмножеств решить задачу наименьших квадратов одним из известных способов, а затем сравнить значения целевых функций и выбрать окончательное решение задачи. Такой подход на практике не реализуем, так как число таких подзадач растёт как степень двойки размерности пространства параметров.

Вместо этого, заметим, что целевая функция (4.22) — выпуклая, допустимое множество X, заданное условиями (4.23), — выпуклое, следовательно, условие (4.77) является необходимым и достаточным условием оптимальности ограниченной задачи. Перепишем (4.77) для конкретного вида операции проекции и градиента целевой функции:

$$\left(\mathbf{x}^*\right)_i = \max\left\{0, \left(\mathbf{x}^* + \lambda A^T(\hat{\mathbf{b}} - A\mathbf{x}^*)\right)_i\right\}, \tag{4.79}$$

откуда видно, что либо одновременно $\left(\mathbf{x}^*\right)_i = 0$ и соответствующий компонент вектора антиградиента $\left(A^T(\hat{\mathbf{b}} - A\mathbf{x}^*)\right)_i \leq 0$, либо $\left(\mathbf{x}^*\right)_i > 0$ и соответствующий компонент вектора градиента равен нулю.

Предлагаемый алгоритм строит последовательность $\mathbf{x}^{(k)}$, начинающуюся с $\mathbf{x}^{(0)} = 0$ и сходящуюся к решению задачи $\mathbf{x}^*$, а также дополнительно хранит $\mathcal{Z}^{(k)}$ — множество индексов таких, что $\left(\mathbf{x}^{(k)}\right)_i = 0$ для всех $i \in \mathcal{Z}^{(k)}$, и дополнительное к этому множеству $\mathcal{P}^{(k)}$. Обозначение $A_{\mathcal{P}}$ используется для редуцированной матрицы задачи, построенной только из тех колонок исходной матрицы A, индексы которых принадлежат множеству $\mathcal{P}$.

Алгоритм состоит из основного и вспомогательного циклов:

- Каждый шаг начинается с вычисления вектора антиградиента $\mathbf{w}^{(k)} \equiv A^T(\hat{\mathbf{b}} - A\mathbf{x}^k)$ и проверки условий (4.79) для $i \in \mathcal{Z}^{(k)}$, условия для $i \in \mathcal{P}^{(k)}$ выполняются автоматически по построению каждого приближения.

- Если оказалось, что необходимые и достаточные условия (4.79) не выполнены, т.е. существует такой индекс t, при котором одновременно $x_t^{(k)} = 0$ и $w_t^{(k)} > 0$, тогда необходимо выполнить процедуру поиска лучшего приближения $\mathbf{x}^{(k+1)}$, в котором, очевидно, $x_t^{(k+1)} > 0$ и его значение подлежит определению. В случае, когда условия нарушены сразу для нескольких индексов, разумно выбрать и работать с индексом соответствующим максимальному значению антиградиента: $t = \arg\max_{i \in \mathcal{Z}^{(k)}} w_i^{(k)}$.

[12] *Non-negative least squares (NNLS)*

- Итак, теперь $\mathcal{Z}^{(k+1)}$ состоит из всех индексов $\mathcal{Z}^{(k)}$, кроме t, а в $\mathcal{P}^{(k+1)}$ индекс t, напротив, добавлен. Чтобы вычислить $x_i^{(k+1)}$ для $i \in \mathcal{P}^{(k+1)}$ требуется решить редуцированную задачу наименьших квадратов с матрицей $A_{\mathcal{P}^{(k+1)}}$ одним из уже известных способов, в то время как $x_i^{(k+1)} = 0$ для $i \in \mathcal{Z}^{(k+1)}$.

- Если $x_i^{(k+1)} \geq 0$ для всех индексов i, тогда следует начать новый шаг с проверки условий (4.79), иначе потребуется вспомогательный восстановительный цикл:

  – Рассмотрим отрезок $\mathbf{x} = \lambda \mathbf{x}^{(k+1)} + (1-\lambda)\mathbf{x}^{(k)}$, для $\lambda \in [0,1]$, и найдём $\lambda^{'} > 0$ соответствующую пересечению с границей допустимого множества X. Это удобно сделать перебором:

$$\lambda^{'} = \min_{\substack{i \in \mathcal{P}^{(k+1)} \\ x_i^{(k+1)} \leq 0}} \frac{x_i^{(k)}}{x_i^{(k)} - x_i^{(k+1)}}.$$

    В силу выпуклости целевой функции, точка $\mathbf{x}^{(k+2)} = \lambda' \mathbf{x}^{(k+1)} + (1-\lambda')\mathbf{x}^{(k)}$ обеспечивает приближение с меньшим значение целевой функции, чем в исходной точке $\mathbf{x}^{(k)}$. Множества $\mathcal{Z}^{(k+2)}$ и $\mathcal{P}^{(k+2)}$ определены согласованным с $\mathbf{x}^{(k+2)}$ образом.

  – Заметим теперь, что после процедуры выбора $\lambda^{'}$ компоненты градиента целевой функции в точке $\mathbf{x}^{(k+2)}$ могут быть отличны от нуля для индексов $i \in \mathcal{P}^{(k+2)}$, что противоречит условиям (4.79). Необходимо вновь решить редуцированную задачу наименьших квадратов с матрицей $A_{\mathcal{P}^{(k+2)}}$, что приведёт к новому улучшенному приближению, некоторые из компонентов которого в свою очередь опять могут оказаться отрицательными, и в таком случае вспомогательный цикл следует вновь повторить.

Видно, что на каждом шаге основного и вспомогательного циклов значение целевой функции убывает. Значит, алгоритм сойдётся к решению через конечное число шагов, среднее число которых оценивается авторами в $O(n)$.

4.3.2 Методы штрафных функций и барьерные методы

К сожалению, численное решение задачи математического программирования (4.4)–(4.6) в самом общем её виде затруднительно. *Методы штрафных функций*[13] — одна из идей, которую можно попробовать применить для решения такой задачи. Принцип состоит в том, чтобы заменить ограниченную задачу оптимизации (4.4)–(4.6) на последовательность неограниченных задач оптимизации, решения которых сходятся к решению исходной задачи.

[13] *Penalty-function methods*

В разделе 2.4, кроме всего прочего, обсуждался метод регуляризации Тихонова, который состоял в модификации целевой функции метода линейных наименьших квадратов путём добавления аддитивного члена с некоторым весом α. Было наглядно показано, что по мере $\alpha \to \infty$ искомая оценка $\hat{\theta}_\delta \to \mathbf{0}$ независимо от задачи. Идейно, предлагается поступить похожим образом, т.е. заменить исходную целевую функцию на новую:

$$\Phi(\mathbf{x}, C) \equiv f(\mathbf{x}) + \phi(\mathbf{x}, C), \tag{4.80}$$

где $\phi(\mathbf{x}, C)$ — штрафная функция, а C — некоторый параметр. Штрафная функция $\phi(\mathbf{x}, C) \to 0$ при $C \to \infty$ внутри допустимого множества X, возможно, за исключением границы множества, вместе с этим $\phi(\mathbf{x}, C) \to \infty$ при $C \to \infty$ вне допустимого множества X.

Приведём несколько примеров популярных штрафных функций:

- Степенная штрафная функция:

$$\phi(\mathbf{x}, C) = C \sum_{i \in \mathcal{I}} \left| \min\left\{0, g_i(\mathbf{x})\right\} \right|^q, \quad q > 0. \tag{4.81}$$

- Квадратичная штрафная функция:

$$\phi(\mathbf{x}, C) = C \sum_{i \in \mathcal{E}} g_i^2(\mathbf{x}). \tag{4.82}$$

- Экспоненциальная штрафная функция:

$$\phi(\mathbf{x}, C) = C \sum_{i \in \mathcal{I}} \exp\left(-C g_i(\mathbf{x})\right). \tag{4.83}$$

- Обратная функция:

$$\phi(\mathbf{x}, C) = \begin{cases} \frac{1}{C} \sum\limits_{i \in \mathcal{I}} g_i^{-1}(\mathbf{x}), & \text{если } g_i(\mathbf{x}) > 0 \quad \forall i \in \mathcal{I} \\ +\infty & \text{иначе.} \end{cases} \tag{4.84}$$

- Логарифмическая функция:

$$\phi(\mathbf{x}, C) = -\frac{1}{C} \sum_{i \in \mathcal{I}} \ln g_i(\mathbf{x}). \tag{4.85}$$

Функции вида (4.84) или (4.85) часто называют *барьерными*[14] функциями, так как такие функции, в отличии например от (4.81), отличны от нуля внутри допустимой области и начинают сильно расти по мере приближения к границе.

[14] *Barrier*

Может возникнуть наивное желание, состоящее в том, чтобы выбрать C достаточно большим числом и применить один из известных алгоритмов численного решения неограниченной задачи оптимизации для минимизации функции (4.80). Отметим, что на практике так сделать невозможно, так как это непрактично с численной точки зрения из-за возникающей овражности функции $\Phi(\mathbf{x}, C)$, больших значений градиента, и резких перепадов функции. Вместо этого, следует рассматривать последовательность задач оптимизации (4.80) для $C_k \to \infty$, находя на каждом шаге $\mathbf{x}^{(k)}$ — минимум задачи соответствующей C_k. Сделать это можно каким-то методом безусловной численной оптимизации. Если повторять этот шаг, одновременно увеличивая C_k и уменьшая ошибку ϵ_k, то последовательность $\mathbf{x}^{(k)}$ будет стремиться к локальному решению исходной ограниченной задачи.

4.4 Стохастические алгоритмы оптимизации

Термин *стохастическая оптимизация* зачастую понимается двояко. Во-первых, речь может идти об оптимизации случайной целевой функции, т.е. о задаче поиска минимума случайной функции в среднем или наиболее вероятном смысле при условии, что известен набор реализаций этой функции при разных значениях переменной. Можно было бы подумать, что из-за ошибок округления о любой функции можно подумать как о стохастической, но на самом деле имеются ввиду задачи близкие области, которая называется теория управления.

Во-вторых, речь может идти о том, что в алгоритм вычисления улучшенного приближения $\mathbf{x}^{(k+1)} = \mathbf{S}_k\left[\mathbf{x}^{(k)}\right]$ вносится элемент случайности. Мы будем использовать термин стохастическая оптимизация только в этом смысле и говорить о *стохастических алгоритмах оптимизации*. В отличии от детерминированных алгоритмов численной оптимизации, теперь идёт речь о случайных последовательностях $\mathbf{x}^{(k)}$, которые, как ожидается, сходятся в среднем или с некоторой вероятностью к критической точке целевой функции. В качестве меры эффективности алгоритма, с одной стороны, аналогично детерминированному случаю мы можем рассматривать асимптотическое поведение среднего уклонения точки $\mathbf{x}^{(k)}$ от критической точки $\mathbf{x}^{(*)}$ с ростом номера шага. С другой стороны, сходимость стохастических алгоритмов оптимизации часто гарантируется с некоторой конечной вероятностью p, которая некоторым образом может зависеть как от настоечных параметров, так и от ожидаемой точности решения или числа итераций.

Приведём пример простого стохастического алгоритма оптимизации, известного как *метод случайного спуска*. Метод состоит из двух шагов:

1. Пусть $\mathbf{x}^{(k)}$ текущая точка, выберем случайную точку $\mathbf{x}'$ в окрестности $\|\mathbf{x}^{(k)} - \mathbf{x}'\|^2 \leq \Delta$.

2. Если $f(\mathbf{x}') < f(\mathbf{x}^{(k)})$, то $\mathbf{x}^{(k+1)} = \mathbf{x}'$, иначе выберем другую реализацию $\mathbf{x}'$ и повторим сравнение.

В эффективности такого подхода возникают естественные сомнения, так как при неудачном выборе распределения $\mathbf{x}'$ вероятность события $f(\mathbf{x}') < f(\mathbf{x}^{(k)})$ может оказаться малой.

4.4.1 Стохастический градиентный спуск

Стохастический градиентный спуск[15] — одни из важных представителей стохастических алгоритмов оптимизации, завоевавший популярность в последнее время благодаря росту популярности методов машинного обучения. Считается, что впервые идея метода была опубликована в работе Robbins и Monro 1951.

Метод применим для оптимизации целевых функций, представимых в виде суммы:

$$f(\mathbf{x}) = \sum_{i=1}^{m} f_i(\mathbf{x}), \tag{4.86}$$

и, следовательно,

$$\nabla f(\mathbf{x}) = \nabla \sum_{i=1}^{m} f_i(\mathbf{x}). \tag{4.87}$$

Например, таким свойством очевидно обладает целевая функция задачи нелинейных наименьших квадратов (4.24).

Ранее обсуждалась дилемма состоящая в том, что на каждом шаге алгоритма требуется вычислить аппроксимацию целевой функции и баланс между качеством этой аппроксимации и требуемыми вычислительными ресурсами не всегда очевиден. Так, например, можно затратить вычислительные ресурсы для расчёта матрицы вторых производных целевой функции, и достичь более высокой скорости сходимости и меньшего числа требуемых шагов, либо применить алгоритм требующий расчёта только первых производных, который потребует большего числа шагов, зато каждый шаг будет рассчитываться очень быстро. В задачах типа (4.24) число m может оказаться относительно велико, например, когда речь идёт о применении метода нейронных сетей к задаче регрессии (4.24), то размер так называемой обучающей выборки варьируется от тысяч до миллионов записей, каждая из которых является отдельным элементом наблюдаемого вектора $\hat{\mathbf{b}}$. В таком случае, вычисление даже самой целевой функции $f(\mathbf{x})$ и её градиента $\nabla f(\mathbf{x})$ требует заметных вычислительных ресурсов.

Идея метода стохастического градиентного спуска состоит в следующем. Заметим, что задача минимизации (4.86) эквивалентна задаче

$$f(\mathbf{x}) = \frac{1}{m} \sum_{i=1}^{m} f_i'(\mathbf{x}), \tag{4.88}$$

где правая часть выражения имеет вид выборочного среднего, составленного из m элементов. При этом, если рассмотреть такую же сумму по любому

[15] *stochastic gradient descent*

подмножеству индексов i, то получится тоже выборочное среднее, обладающее меньшей точностью в силу меньшего размера выборки. Понятно, что для метода градиентного спуска вместо $\nabla f(\mathbf{x}^{(k)})$ можно взять

$$\mathbf{g}^{(k)} \equiv \sum_{i \in I^{(k)}} \nabla f_i(\mathbf{x}), \tag{4.89}$$

где $I^{(k)}$ некоторое случайное подмножество индексов маленького размера, новое на каждом шаге, часто его называют *пакетом*[16]:

$$\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)} - \alpha_k \mathbf{g}^{(k)}. \tag{4.90}$$

При таком подходе скорость вычисления каждого шага может быть увеличена относительно детерминированного метода градиентного спуска.

4.4.2 Метод адаптивной оценки моментов

Понятно, что идея стохастического градиентного спуска имеет множество дополнений и улучшений. Например, предложенный в работе Kingma и Ba 2017, метод адаптивной оценки моментов[17] улучшает точность с которой оценивается вектор градиента $\nabla f_i(\mathbf{x})$. Понятно, что среднее различие между $\nabla f(\mathbf{x}^{(k)})$ и $\mathbf{g}^{(k)}$ увеличивается по мере уменьшения размера выборки $I^{(k)}$, а увеличивать эту выборку мы не хотим по причинам вычислительных затрат. Вместо этого, можно в некотором смысле заменить среднее по ансамблю средним по времени, предполагая, что на шагах k и $k+1$ вектора $\nabla f(\mathbf{x}^{(k)})$ и $\nabla f(\mathbf{x}^{(k+1)})$ отличаются не сильно, т.е. много меньше, чем неопределённость вносимая за счёт стохастического метода оценивания.

Предлагается наряду с вектором $\mathbf{g}^{(k)}$ вычислять и хранить его сглаженную версию $\mathbf{m}^{(k)}$:

$$\mathbf{m}^{(k)} = \beta_1 \mathbf{m}^{(k-1)} + (1 - \beta_1)\mathbf{g}^{(k)}, \tag{4.91}$$

где β_1 — константа сглаживания, например $\beta_1 = 0.9$, и вектор оценки поэлементных нецентральных вторых моментов $\mathbf{v}^{(k)}$:

$$v_i^{(k)} = \beta_2 v_i^{(k-1)} + (1 - \beta_2) \cdot \left(g_i^{(k)}\right)^2, \tag{4.92}$$

где β_2 — константа сглаживания для вектора $\mathbf{v}^{(k)}$, например, равная $\beta_2 = 0.999$. Вектор $\mathbf{v}^{(k)}$ характеризует степень разброса каждой компоненты $\mathbf{g}^{(k)}$. Следующее приближение вычисляется аналогично методу стохастического градиентного спуска:

$$x_i^{(k+1)} = x_i^{(k)} - \frac{\alpha}{1 - \beta_1^k} \left(\epsilon + \sqrt{\frac{v_i^{(k)}}{1 - \beta_2^k}} \right)^{-1} m_i^{(k)}, \tag{4.93}$$

где рекомендуемые значения констант $\alpha = 0.001$ и $\epsilon = 10^{-8}$.

[16] *batch*

[17] Adaptive moment estimator (ADAM)

4.4.3 Методы Монте-Карло по схеме марковской цепи

Алгоритм имитации отжига[18] (имеется ввиду металлургический термин) — ещё один популярный стохастический алгоритм численной оптимизации, позволяющий находить глобальный минимум функции и обладающий красивой физической интерпретацией. Однако, перед рассмотрением непосредственно задачи оптимизации, изучим вспомогательную задачу о генерации реализаций случайного вектора с требуемым распределением.

Алгоритм Метрополиса

Николас Метрополис вместе с коллегами по Лос-Аламосской национальной лаборатории предложили (см. Metropolis и др. 1953) численный метод взятия многомерных интегралов из статистической физики:

$$\langle X \rangle = \int \mathbf{dpdq} X(\mathbf{p},\mathbf{q}) \exp\left(-\frac{E(\mathbf{p},\mathbf{q})}{\theta}\right), \tag{4.94}$$

где X некоторая физическая величина, которую требуется усреднить по ансамблю большого числа частиц, а вектора $\mathbf{p} \in R^{3n}$ и $\mathbf{q} \in R^{3n}$ обозначают состояние системы в фазовом пространстве, θ — некоторый параметр, называемый температурой, и $E(\mathbf{p},\mathbf{q})$ — обобщённая энергия системы. Понятно, что проблема размерности помешает взять такой интеграл классическими методами на равномерной сетке, однако и наивный подход Монте-Карло с набрасыванием точек будет не очень результативен, так как экспонента стоящая в подынтегральном выражении принимает значения близкие к нулю почти во всем фазовом пространстве. Выход состоит в том, чтобы генерировать случайные состояния системы $(\mathbf{p},\ \mathbf{q})$ таким образом, чтобы плотность вероятности значений функции $E(\mathbf{p},\mathbf{q})$ была пропорционально $\exp\left(-\frac{E(\mathbf{p},\mathbf{q})}{\theta}\right)$.

Пусть известно текущее состояние $\mathbf{x}^{(k)} \equiv (\mathbf{p}^{(k)},\mathbf{q}^{(k)})$, сгенерируем новое пробное состояние $\mathbf{x}'$ используя вспомогательное условное распределение $p_G(\mathbf{x}'|\mathbf{x}^{(k)})$. Вспомогательное распределение $p_G(\mathbf{x}'|\mathbf{x}^{(k)})$ предполагается таким, для которого известен эффективный алгоритм генерирования реализаций. Например, в оригинальной работе предлагается следующая схема:

$$\mathbf{x}' = \mathbf{x}^{(k)} + \alpha\xi, \tag{4.95}$$

где α параметр, а ξ вектор случайных величин равномерно распределённых от -1 до 1. В качестве альтернативы мы могли бы предложить использовать многомерное нормальное распределение с средним в точке $\mathbf{x}^{(k)}$ и некоторой дисперсией σ_0^2 одинаковой для всех компонент. Сравним значения функции $E(\mathbf{x}')$ и $E(\mathbf{x}^{(k)})$:

- Если $E(\mathbf{x}') < E(\mathbf{x}^{(k)})$, то $\mathbf{x}^{(k+1)} = \mathbf{x}'$

[18] *simulated annealing*

- Если $E(\mathbf{x}') \geq E(\mathbf{x}^{(k)})$, тогда
  - $\mathbf{x}^{(k+1)} = \mathbf{x}'$ с вероятностью $\exp\left(-\frac{\Delta E}{\theta}\right)$,
  - $\mathbf{x}^{(k+1)} = \mathbf{x}^{(k)}$ с вероятностью $1 - \exp\left(-\frac{\Delta E}{\theta}\right)$.

Во втором случае разыгрывается дополнительная случайная величина, равномерно распределённая на отрезке $[0, 1]$, и на основании её значения и значения $\exp\left(-\frac{\Delta E}{\theta}\right)$ происходит окончательное решение.

Видно, что представленный алгоритм фактически задаёт способ построения марковской цепи, причём вероятность перехода из состояния i в состояние $j \neq i$ определяется как

$$p_{ij} \equiv p(\mathbf{x}^{(k+1)} = \mathbf{x}_j | \mathbf{x}^{(k)} = \mathbf{x}_i) = p_G(\mathbf{x}^{(k+1)} | \mathbf{x}^{(k)}) \cdot \min\left\{1, \exp\left(-\frac{\Delta E}{\theta}\right)\right\}, \tag{4.96}$$

при этом выполняется условие нормировки вероятности

$$\sum_j p_{ij} = 1, \tag{4.97}$$

откуда при необходимости можно определить вероятность остаться в текущем состоянии $p_{ii} = p(\mathbf{x}^{(k+1)} = \mathbf{x}_i | \mathbf{x}^{(k)} = \mathbf{x}_i)$. Для простоты, мы считаем, что число возможных состояний дискретно, но для континуума состояний рассуждения тоже остаются верными, а соответствующие суммы заменяются на интегралы.

Видно, что вероятность перехода из состояния i в состояние j не зависит от номера шага k, значит марковская цепь является однородной по времени. Одним из важных свойством алгоритма должна быть возможность достигнуть любое состояние из любого другого за конечное число шагов с ненулевой вероятностью, это достигается правильным выбором $p_G(\mathbf{x}^{(k+1)} | \mathbf{x}^{(k)})$. Известно, что хорошая цепь Маркова будет иметь предельное распределение вероятностей состояний для $k \to \infty$, а в случае эргодической цепи можно сказать, что частота разных состояний во времени соответствует предельному распределению вероятностей:

$$p(\mathbf{x}^{(k)}) = p(\mathbf{x}^{(\infty)}). \tag{4.98}$$

Используя принцип детального равновесия можно показать, что в случае алгоритма Метрополиса вероятность выпадения каждого состояния i действительно пропорциональна $\exp\left(-\frac{\Delta E}{\theta}\right)$. Пусть N_i — число точек в состоянии i, которое нам встретилось за некоторое большое число шагов $N \to \infty$, пусть N_j — число точек в состоянии j, и пусть для определённости $E(\mathbf{x}_i) < E(\mathbf{x}_j)$. Согласно принципу детального равновесия, в эргодической цепи Маркова количество переходов «туда» (из i и j) должно соответствовать числу переходов «оттуда» (из j в i). Количество переходов из i в j равно $N_i \exp\left(-\frac{E(\mathbf{x}_j) - E(\mathbf{x}_i)}{\theta}\right)$, количество обратных переходов — строго N_j, так как в состоянии i значение энергии меньше, чем в состоянии j. Приравнивая

число переходов между состояниями в обе стороны, получим следующую пропорцию:

$$\frac{N_i}{N_j} = \frac{\exp\left(-\frac{E(\mathbf{x}_i)}{\theta}\right)}{\exp\left(-\frac{E(\mathbf{x}_j)}{\theta}\right)}, \tag{4.99}$$

откуда понятно, что $p_i = \frac{N_i}{N} \sim \exp\left(-\frac{E(\mathbf{x}_i)}{\theta}\right)$.

Алгоритм имитации отжига

Из курса статистической физики известна теорема, именуемая иногда третьим началом термодинамики. Распределение $p(\mathbf{x}) \sim \exp\left(-\frac{E(\mathbf{x})}{\theta}\right)$ при $\theta \to 0$ устремится к дельта-функции в точке $\mathbf{x}^*$, соответствующей глобальному минимуму функции $E(\mathbf{x})$. Иными словами, при температуре абсолютного нуля система многих частиц может занимать только одно состояние — соответствующее минимально возможной энергии.

Используя целевую функцию $f(\mathbf{x})$ вместо $E(\mathbf{x})$ можно попытаться построить распределение $p(\mathbf{x})$ при очень маленьком значении параметра θ. Однако, чем меньше θ тем больше шагов требуется для достижения равновесного состояния, поэтому на практике применяется подход постепенного охлаждения $\theta_k \to 0$, который и называется алгоритмом имитации отжига. Подход в чём-то похож на подход из раздела 4.3.2 о методах штрафных функций.

При больших температурах θ (начальную температуру следует выбрать такой, чтобы разрешить системе занимать все возможные состояния с большой вероятностью) система быстро приходит в равновесное состояние, распределение в котором $p(\mathbf{x}) \sim \exp\left(-\frac{E(\mathbf{x})}{\theta}\right)$. При постепенном и правильном уменьшении параметра θ распределение будет квазиравновестно сходиться к ожидаемому распределению в окрестности решения $\mathbf{x}^*$.

К сожалению, снизить температуру не испортив равновесности состояния бывает достаточно затруднительно. Предлагаются различные схемы охлаждения, например

$$\theta^{(k+1)} = \alpha\theta^{(k)}, \tag{4.100}$$

где α параметр, который чуть меньше единицы. Рекомендуется делать понижение температуры следует делать раз в некоторое число итераций основного алгоритма Метрополиса. Ориентиром может являться число принятых переходов, свидетельствующее о том, что система пришла в равновесие. Завершать алгоритм предлагается в тот момент, когда отношение принятых и предложенных переходов становится меньше некоторой границы.

Следует однако сказать, что при слишком консервативном (медленном) охлаждении алгоритм сходится чрезвычайно медленно, а при быстром охлаждении алгоритм попадает в локальные минимумы (аналог метастабильных состояний в кристаллических решётках) и застревает там.

Глава 5

Метод максимального правдоподобия

В предыдущих главах мы изучали задачу оценки неизвестных параметров модели при известных, пусть и зашумленных, результатах измерений тех величин, которые модель, собственно, и предсказывает. Такая задача сводится к поиску параметров, при которых модель наилучшим образом предсказывает измеренные значения.

Теперь рассматривается другая задача. Пусть наши измерения — реализации какой-то случайной величины, или случайного вектора. Будем считать, что случайность в данном случае — свойство присущее измеряемой физической величине. Конечно, в самом простом случае, случайность обусловлена шумами и погрешностями измерений. Но это не единственный источник, например, мы можем регистрировать энергию частиц, возникающих при некотором распаде, и даже с учётом конечной точности измерений окажется, что каждая конкретная частица имеет своё значение энергии, а вся совокупность имеет некоторое распределение.

Допустим мы знаем класс распределений, но хотели бы оценить конкретные параметры распределения, либо нам дополнительно известна модель, выражающая параметры распределения через другие параметры, имеющие самостоятельный смысл. *Метод максимального правдоподобия*[1] — один из способов определения параметров распределения случайной величины по известной выборке реализаций этой величины. Его идея состоит в следующем. Представим, что нам известна реализация некоторого случайного вектора $\mathbf{x}$ и известна функция плотности вероятности этого случайного вектора $p(\mathbf{x}|\theta)$, с точностью до некоторых параметров распределения θ, которые мы и хотим оценить. *Функцией правдоподобия*[2] называется плотность вероятности случайного вектора, вычисленная для известной реализации:

$$L(\theta) \equiv p(\mathbf{x}|\theta). \tag{5.1}$$

[1] *Maximum likelihood estimation*
[2] *likelihood*

Предположим, что раз мы увидели реализацию $\mathbf{x}$, то для истинных параметров θ вероятность такой реализации $p(\mathbf{x}|\theta)$ должна быть максимально возможной, среди всех альтернатив.

Конечно, максимальная вероятность исхода не гарантирует, что этот исход будет случаться всегда, но интуитивно понятно, что следуя этому предположению мы будем ошибаться реже всего. Метод максимального правдоподобия состоит в максимизации функции правдоподобия относительно неизвестных параметров распределения θ:

$$\hat{\theta} = \arg\max_{\theta} L(\theta) = \arg\max_{\theta} \ln L(\theta). \tag{5.2}$$

На практике вместо максимизации функции правдоподобия $L(\theta)$ удобнее использовать её логарифм $\ln L(\theta)$. Во-первых, логарифм в силу своей монотонности не изменит положения экстремумов. Во-вторых, $\ln L(\theta)$ упрощает выражения для функций плотностей вероятности, имеющих экспоненциальную зависимость. В-третьих, оказывается удобным рассматривать важный частный случай выборки независимых одинаково распределённых величин. Например, пусть известно N реализаций случайного вектора $\mathbf{x}_i \in \mathrm{R}^{\mathrm{n}}$, тогда

$$\ln L(\theta) = \ln p(\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N|\theta) = \ln \prod_{i=1}^{N} p(\mathbf{x}_i|\theta) = \sum_{i=1}^{N} \ln p(\mathbf{x}_i|\theta). \tag{5.3}$$

Рассмотрим некоторые простые примеры, чтобы оценить насколько разумными получаются оценки параметров $\hat{\theta}$. Пусть дана выборка из N одинаково распределённых случайных величин x_i с нормальным распределением, найдём оценки параметров μ и σ^2. Для этого запишем логарифм функции правдоподобия

$$\ln L(\theta) = \sum_{i=1}^{N} \left(-\frac{1}{2} \ln 2\pi\sigma^2 - \frac{1}{2\sigma^2} (x_i - \mu)^2 \right), \tag{5.4}$$

и, дифференцируя выражение по μ и σ^2, окончательно находим

$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^{N} x_i, \tag{5.5}$$

$$\hat{\sigma}^2 = \frac{1}{N} \sum_{i=1}^{N} (x_i - \hat{\mu})^2 . \tag{5.6}$$

Предполагается, что сначала вычисляется оценка среднего $\hat{\mu}$ из (5.5), а затем она может быть подставлена как число в (5.6). Обратим внимание, что оценка параметра $\hat{\sigma}^2$, получаемая по методу максимального правдоподобия, оказывается смещённой.

Аналогично, пусть дана выборка из N одинаково распределённых случайных величин x_i с экспоненциальным распределением

$$p(x_i|\lambda) = \lambda \exp(-\lambda x_i), \tag{5.7}$$

найдём оценку параметра этого распределения. Логарифм функции правдоподобия будет выглядеть следующим образом:

$$\ln L(\theta) = \sum_{i=1}^{N} (\ln \lambda - \lambda x_i), \tag{5.8}$$

дифференцируя его по λ находим, что

$$\hat{\lambda} = \frac{N}{\sum_{i=1}^{N} x_i}. \tag{5.9}$$

Теперь рассмотрим более сложный пример, демонстрирующий, что метод максимального правдоподобия способен приводить к менее тривиальным результатам. Пусть у нас как и в разделе 2 есть линейная модель $A\theta = \mathbf{b}$. Как и прежде, мы считаем, что нам доступны отягощённые погрешностью ϵ измерения $\hat{\mathbf{b}} = \mathbf{b} + \epsilon$. Предположим, что все ϵ_i независимы и распределены нормально с нулевым средним и одинаковой дисперсией σ^2, тогда логарифм функции правдоподобия вектора $\hat{\mathbf{b}}$ выражается следующим образом:

$$\ln L(\hat{\mathbf{b}}) = -\frac{n}{2} \ln 2\pi\sigma^2 - \frac{1}{2\sigma^2} \|\hat{\mathbf{b}} - A\theta\|^2, \tag{5.10}$$

где подлежащими оценке параметрами распределения являются константа σ^2 и вектор $\theta \in \mathrm{R}^{\mathrm{m}}$. Дифференцируя по параметрам, получим:

$$A^T A\hat{\theta} = A^T \hat{\mathbf{b}}, \tag{5.11}$$

$$\hat{\sigma}^2 = \frac{1}{n} \|\hat{\mathbf{b}} - A\hat{\theta}\|^2 = \frac{1}{n} \|\hat{\epsilon}\|^2. \tag{5.12}$$

Сравнив результат с формулами (2.10) и (2.17), можно увидеть, что при определённых предположениях метод наименьших квадратов может быть получен из метода максимального правдоподобия как частный случай. Оценка дисперсии по формуле (5.12) имеет смещение в отличии от (2.17).

Рассмотрим несколько полезных примеров, относящихся к многомерному нормальному распределению. Пусть в нашем распоряжении N независимых реализаций $\mathbf{x}_i \in \mathrm{R}^{\mathrm{m}}$, каждая из которых распределена нормально со средним μ и матрицей ковариации $\Sigma(\alpha)$, которая зависит от некоторого параметра α. Логарифм функции правдоподобия записывается как

$$\ln L(\theta) = -\frac{Nm}{2} \ln 2\pi - \frac{N}{2} \ln \det \Sigma(\alpha) - \frac{1}{2} \sum_{i=1}^{N} (\mathbf{x}_i - \mu, \Sigma^{-1}(\alpha) (\mathbf{x}_i - \mu)). \tag{5.13}$$

Используя градиент по вектору μ получаем выражение

$$\mu = \frac{1}{N} \sum_{i=1}^{N} \mathbf{x}_i. \tag{5.14}$$

Возьмём производную по параметру α, и после некоторых манипуляций получим:

$$\operatorname{tr}\Sigma^{-1}(\hat{\alpha})\left.\frac{\partial\Sigma(\alpha)}{\partial\alpha}\right|_{\hat{\alpha}}\left(I-\Sigma^{-1}(\hat{\alpha})\frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i-\hat{\mu})\otimes(\mathbf{x}_i-\hat{\mu})\right)=0, \tag{5.15}$$

где символ $\otimes$ обозначает внешнее векторное произведение. Здесь было использовано общее свойство

$$\frac{\partial\det\Sigma(\alpha)}{\partial\alpha}=\det\Sigma(\alpha)\operatorname{tr}\left(\Sigma^{-1}(\alpha)\frac{\partial\Sigma(\alpha)}{\partial\alpha}\right), \tag{5.16}$$

известное как формула Якоби.

Выражение (5.15) упрощается, если ввести дополнительные предположения. Например, пусть α — поочерёдно каждый элемент матрицы Σ, тогда

$$\hat{\Sigma}=\frac{1}{N}\sum_{i=1}^{N}(\mathbf{x}_i-\hat{\mu})\otimes(\mathbf{x}_i-\hat{\mu}). \tag{5.17}$$

Пусть $\Sigma=\sigma^2 I$, и параметр $\alpha=\sigma^2$, тогда

$$\hat{\sigma}^2=\frac{1}{Nd}\sum_{i}\|\mathbf{x}_i-\mu\|^2. \tag{5.18}$$

Пусть Σ блочная диагональная матрица:

$$\Sigma=\begin{bmatrix}\Sigma_1 & 0\\ 0 & \Sigma_2\end{bmatrix},$$

тогда можно показать, что уравнение (5.15) справедливо отдельно для Σ_1 и для Σ_2, причём от внешнего произведения нужно взять соответствующий блок нужного размера.

5.1 Модели со скрытыми переменными и EM-алгоритм

Не всегда логарифм функции правдоподобия предполагает форму, удобную для аналитического поиска максимума. В частности, так происходит в случае моделей со *скрытыми переменными*. Несмотря на то, что мы могли бы применить один из методов численной оптимизации, существует ещё более мощный и изящный метод, предложенный американскими исследователями (см. Dempster, Laird и Rubin 1977), который позволяет применять метод максимального правдоподобия удобно и эффективно в этих случаях.

Скрытой переменной принято называть физическую величину, которая непосредственно в измерениях не регистрируется, но в то же время и не является искомым параметров модели. Например, мы измеряем энергию

частиц некоторым детектором: в таком случае измеряемой величиной является значение энергии E[3], предположим ещё, что в ходе нашего эксперимента на детектор попадают частицы двух сортов (это могут быть электроны и мюоны; либо, например, фотоэлектроны и термоэлектроны), проблема в том, что теперь мы получаем смесь распределения энергий для двух сортов частиц. В этом случае мы можем называть тип частицы скрытой переменной, физически она существует, но в ходе эксперимента не измеряется. Предположим, что нет технической возможности полностью подавить поток того или иного рода частиц, тогда мы привлекаем метод максимального правдоподобия, чтобы определить параметры смешанного распределения.

Хотя скрытая переменная не обязана быть дискретной переменной, EM-алгоритм традиционно демонстрируется на примере смеси двух нормальных распределений. Пусть у нас есть скрытая дискретная случайная переменная z, которая принимает значение 0 с вероятностью τ, и, соответственно, значение 1 с вероятностью $1-\tau$, а также пусть x — непрерывная измеряемая нормально распределённая величина. Соответствующие условные функции плотности вероятности запишутся следующим образом:

$$p(x|\{z=0\},\theta) = \frac{1}{\sqrt{2\pi\sigma_1^2}}\exp\left(-\frac{(x-\mu_1)^2}{2\sigma_1^2}\right), \tag{5.19}$$

$$p(x|\{z=1\},\theta) = \frac{1}{\sqrt{2\pi\sigma_2^2}}\exp\left(-\frac{(x-\mu_2)^2}{2\sigma_2^2}\right). \tag{5.20}$$

Пусть мы располагаем выборкой из N независимых реализаций x_i, и интересующий нас набор параметров состоит из пяти величин: τ, μ_1, μ_2, σ_1^2, σ_2^2.

Так как величина z_i является скрытой, для применения метода максимального правдоподобия необходимо вычислить маргинальное распределение для x_i:

$$\begin{aligned} p(x_i|\theta) &= \sum_{j\in\{0,1\}} p(x_i|\{z_i=j\},\theta)p(\{z_i=j\}|\theta) = \\ &= \frac{\tau}{\sqrt{2\pi\sigma_1^2}}\exp\left(-\frac{(x_i-\mu_1)^2}{2\sigma_1^2}\right) + \frac{1-\tau}{\sqrt{2\pi\sigma_2^2}}\exp\left(-\frac{(x_i-\mu_2)^2}{2\sigma_2^2}\right). \end{aligned} \tag{5.21}$$

Логарифм правдоподобия для всей выборки измерений по прежнему равен следующей сумме:

$$\ln L(\theta) = \sum_{i=1}^{N} \ln p(x_i|\theta). \tag{5.22}$$

Мы видим, что структура этой функции неудобна для аналитического поиска максимума, функция содержит в себе сумму логарифмов от суммы

[3]Чаще какой-то пропорциональной энергии величины, например заряда или напряжения.

двух слагаемых. Оценки параметров θ не только не будут иметь аналитического вида, но их поиск также будет затруднён на практике с помощью численных методов оптимизации.

Итак, EM-алгоритм (метод) состоит из двух последовательных шагов, повторяющихся итеративно до достижения результата. Первый шаг — *усреднение*[4], второй шаг — *максимизация*[5]. Ключевой идеей метода является рассмотрение среднего логарифма полного правдоподобия (как если бы скрытая переменная регистрировалась в эксперименте), вместо логарифма маргинального правдоподобия, имеющего неудобную форму. В нашем случае, логарифм полного правдоподобия записывается следующим образом:

$$\ln p(\mathbf{x}, \mathbf{z}|\theta) = \sum_{i=1}^{N} \left((1 - z_i) \ln \tau p(x_i | \{z_i = 0\}, \theta) + z_i \ln(1 - \tau) p(x_i | \{z_i = 1\}, \theta) \right), \tag{5.23}$$

где условные функции плотности вероятности задаются формулами (5.19)–(5.20).

Предположим, что некоторая грубая оценка параметров θ' нам известна (в реальности это оценка параметров выполненная на предыдущем шаге), тогда мы можем оценить значение скрытой переменной z_i для каждого измерения, понятно, что это значение будет зависит от измеренного значения x_i. Усредним логарифм полного правдоподобия по всем скрытым переменным:

$$Q(\theta|\theta') = \mathrm{E}_{\mathbf{z}|\mathbf{x},\theta'} \left[\ln p(\mathbf{x}, \mathbf{z}|\theta) \right]. \tag{5.24}$$

Или в рассматриваемом частном случае смеси двух нормальных распределений:

$$\begin{aligned} Q(\theta|\theta') = \sum_{i=1}^{N} \sum_{j \in \{0,1\}} & p(\{z_i = j\} | x_i, \theta') \left((1 - j) \ln \tau p(x_i | \{z_i = 0\}, \theta) \right. \\ & \left. + j \ln(1 - \tau) p(x_i | \{z_i = 1\}, \theta) \right) = \\ = \sum_{i=1}^{N} & \left(p(\{z_i = 0\} | x_i, \theta') \ln \tau p(x_i | \{z_i = 0\}, \theta) \right. \\ & \left. + p(\{z_i = 1\} | x_i, \theta') \ln(1 - \tau) p(x_i | \{z_i = 1\}, \theta) \right). \end{aligned} \tag{5.25}$$

Вместо логарифма суммы получилась сумма логарифмов, а значит с этим выражением можно удобно работать. Найдём новую улучшенную оценку искомых параметров максимизируя средний логарифм правдоподобия:

$$\hat{\theta} = \arg\max_{\theta} Q(\theta|\theta'). \tag{5.26}$$

[4] expectation
[5] maximization

Выражения вида $p(\{z_i = j\}\,|x_i, \theta')$, встречающиеся в (5.25), могут быть предварительно доведены до числа с использованием известного соотношения:

$$p(\{z_i = j\}\,|x_i, \theta') = \frac{p(x_i|\,\{z_i = j\}\,, \theta')p(\{z_i = j\}\,|\theta')}{\sum_{k\in\{0,1\}} p(x_i|\,\{z_i = k\}\,, \theta')p(\{z_i = k\}\,|\theta')}, \tag{5.27}$$

а значит рассматриваются нами как набор числовых коэффициентов, стоящих перед логарифмами правдоподобий, при дифференцировании (5.25) по искомым параметрам. Подставляя условные функции плотности вероятности (5.19)–(5.20), получим:

$$p(\{z_i = 0\}\,|x_i, \theta') = \frac{\frac{\tau'}{\sqrt{2\pi\sigma_1'^2}}\exp\left(-\frac{(x_i-\mu_1')^2}{2\sigma_1'^2}\right)}{\frac{\tau'}{\sqrt{2\pi\sigma_1'^2}}\exp\left(-\frac{(x_i-\mu_1')^2}{2\sigma_1'^2}\right) + \frac{1-\tau'}{\sqrt{2\pi\sigma_2'^2}}\exp\left(-\frac{(x_i-\mu_2')^2}{2\sigma_2'^2}\right)}, \tag{5.28}$$

$$p(\{z_i = 1\}\,|x_i, \theta') = \frac{\frac{1-\tau'}{\sqrt{2\pi\sigma_2'^2}}\exp\left(-\frac{(x_i-\mu_2')^2}{2\sigma_2'^2}\right)}{\frac{\tau'}{\sqrt{2\pi\sigma_1'^2}}\exp\left(-\frac{(x_i-\mu_1')^2}{2\sigma_1'^2}\right) + \frac{1-\tau'}{\sqrt{2\pi\sigma_2'^2}}\exp\left(-\frac{(x_i-\mu_2')^2}{2\sigma_2'^2}\right)}. \tag{5.29}$$

Подробное выражение для среднего логарифма полного правдоподобия:

$$Q(\theta|\theta') = \text{const} + \sum_{i=1}^{N} p(\{z_i = 0\}\,|x_i, \theta')\ln\tau + \sum_{i=1}^{N} p(\{z_i = 1\}\,|x_i, \theta')\ln(1-\tau) +$$

$$+ \sum_{i=1}^{N} p(\{z_i = 0\}\,|x_i, \theta')\left(-\frac{1}{2}\ln\sigma_1^2\right) - \frac{1}{2\sigma_1^2}\sum_{i=1}^{N} p(\{z_i = 0\}\,|x_i, \theta')(x_i-\mu_1)^2 +$$

$$+ \sum_{i=1}^{N} p(\{z_i = 1\}\,|x_i, \theta')\left(-\frac{1}{2}\ln\sigma_2^2\right) - \frac{1}{2\sigma_2^2}\sum_{i=1}^{N} p(\{z_i = 1\}\,|x_i, \theta')(x_i-\mu_2)^2. \tag{5.30}$$

И, взяв производные по параметрам, получаем простые выражения:

$$\hat{\tau} = \frac{\sum_{i=1}^{N} p(\{z_i = 0\}\,|x_i, \theta')}{N}, \tag{5.31}$$

$$\hat{\mu}_1 = \frac{\sum_{i=1}^{N} p(\{z_i = 0\}\,|x_i, \theta')x_i}{\sum_{i=1}^{N} p(\{z_i = 0\}\,|x_i, \theta')}, \tag{5.32}$$

$$\hat{\sigma}_1^2 = \frac{\sum_{i=1}^{N} p(\{z_i = 0\}\,|x_i, \theta')(x_i - \hat{\mu}_1)^2}{\sum_{i=1}^{N} p(\{z_i = 0\}\,|x_i, \theta')}, \tag{5.33}$$

$$\hat{\mu}_2 = \frac{\sum_{i=1}^{N} p(\{z_i = 1\}\,|x_i, \theta')x_i}{\sum_{i=1}^{N} p(\{z_i = 1\}\,|x_i, \theta')}, \tag{5.34}$$

$$\hat{\sigma}_2^2 = \frac{\sum_{i=1}^{N} p(\{z_i = 1\}\,|x_i, \theta')(x_i - \hat{\mu}_2)^2}{\sum_{i=1}^{N} p(\{z_i = 1\}\,|x_i, \theta')}. \tag{5.35}$$

Сравнивая выражения (5.32) и (5.34) с (5.5) и выражения (5.33) и (5.35) с (5.6), можно заметить, что EM-алгоритм «заменил» выборочное усреднение на взвешенное усреднение, где веса определяют степень принадлежности измерения x_i к тому или иному классу.

Докажем, что предложенный алгоритм сходится к максимуму маргинального правдоподобия. Для этого усредним следующее соотношение с обоих сторон:

$$\ln p(\mathbf{x}|\theta) = \ln p(\mathbf{x}, \mathbf{z}|\theta) - \ln p(\mathbf{z}|\mathbf{x}, \theta), \tag{5.36}$$

получим

$$\ln p(\mathbf{x}|\theta) = Q(\theta|\theta') - H(\theta|\theta'), \tag{5.37}$$

где введено обозначение

$$H(\theta|\theta') \equiv \sum_j p(\{z_i = j\}\,|x_i, \theta') \ln p(\{z_i = j\}\,|x_i, \theta). \tag{5.38}$$

Рассмотрим приращение логарифма маргинального правдоподобия, которое запишется в следующем виде:

$$\ln p(\mathbf{x}|\theta) - \ln p(\mathbf{x}|\theta') = (Q(\theta|\theta') - Q(\theta'|\theta')) + (H(\theta|\theta') - H(\theta'|\theta'))\,. \tag{5.39}$$

Разность $Q(\theta|\theta') - Q(\theta'|\theta') \geq 0$ по построению алгоритма, а разность H:

$$H(\theta|\theta') - H(\theta'|\theta') = -\sum_j p(\{z_i = j\}\,|x_i, \theta') \ln \frac{p(\{z_i = j\}\,|x_i, \theta)}{p(\{z_i = j\}\,|x_i, \theta')}. \tag{5.40}$$

Напомним, что логарифм — функция, выпуклая на интересующем нас интервале, поэтому действует неравенство Йенсона, и правую часть можно

оценить как

$$\sum_j p(\{z_i = j\}\,|x_i, \theta') \ln \frac{p(\{z_i = j\}\,|x_i, \theta)}{p(\{z_i = j\}\,|x_i, \theta')}$$
$$\leq \ln \sum_j p(\{z_i = j\}\,|x_i, \theta') \frac{p(\{z_i = j\}\,|x_i, \theta)}{p(\{z_i = j\}\,|x_i, \theta')} = \ln \sum_j p(\{z_i = j\}\,|x_i, \theta) \leq 0. \tag{5.41}$$

Получается, что

$$\ln p(\mathbf{x}|\theta) - \ln p(\mathbf{x}|\theta') \geq Q(\theta|\theta') - Q(\theta'|\theta'), \tag{5.42}$$

т.е. любое увеличение функции $Q(\theta|\theta')$ приводит к увеличению соответствующего логарифма маргинального правдоподобия, в силу ограниченности последнего алгоритм сходится.

Глава 6

Анализ временных рядов

6.1 Случайные процессы

Ещё одна популярная область применения метода максимального правдоподобия из главы 5 — моделирование и анализ случайных процессов.

Случайные процессы можно считать некоторым обобщением понятия случайной величины. Если случайная величина является отображением $\xi(\omega)$ из пространства элементарных исход в пространство реализаций случайной величины R, то реализацией случайного процесса $\xi(\omega, t)$ является функция некоторая времени $x(t)$. Понятно, что под временем t здесь понимается просто любая скалярная переменная, но для удобства обычно говорят о случайных процессах и времени. Обобщение случайного процесса на случай больших размерностей детерминированного параметра называется случайным полем $\xi(\omega, \mathbf{r})$.

Важным частным случаем случайных процессов являются случайные последовательности, когда время t может принимать только значения на равномерной сетке и заменяется индексом i. С точки зрения физики это означает, что мы обязуемся рассматривать (измерять или предсказывать) значения случайного процесса только на какой-то равномерной сетке по времени. Выделение такого частного случая практически целесообразно, так как позволяет предложить более простые методы анализа данных, с другой стороны равномерные по времени ряды измерений достаточно часто встречаются на практике, особенно с учётом частого применения автоматизации физического эксперимента.

Хотя и подразумевается, что величины $\xi(\omega, t_1)$ и $\xi(\omega, t_2)$ зависимые, в общем случае, случайные процессы нельзя свести к случайным векторам, так как наличие скалярного параметра предполагает возможность продолжать процесс бесконечно в прошлое или будущее, либо бесконечно дробить сетку времён, на которой происходит измерение случайного процесса. Полным и исчерпывающим описанием случайного процесса является многомерная функция плотности вероятности. Для любого N и любого набора $t_1, ..., t_N$

мы обязаны предоставить функцию $p(x_1, t_1, ..., x_N, t_N)$, показывающую совместную плотность распределения измерений значения случайного процесса в заданный набор времён. Стоит отметить, что нужно уметь задавать (пусть не в виде формулы, но хотя бы в смысле эффективного алгоритма позволяющего рассчитать значение) функцию переменного числа аргументов, а это может быть не просто.

Однако, если можно предложить эффективный способ построения такой функции, т.е. решения прямой задачи, то затем, во-первых, можно применить метод максимального правдоподобия для оценки неизвестных параметров многомерной функции плотности вероятности исходя из известной заданной реализации случайного процесса. Во-вторых, используя найденные оценки параметров и известную реализацию случайного процесса, можно решать задачу прогнозирования, т.е. задания плотности вероятности для прошлого, будущего, либо любых других моментов времени, когда нет измерений.

Если многоточечная функция распределения подчиняется свойству сдвига по времени $p(x_1, t_1, ..., x_N, t_N) = p(x_1, t_1 + \Delta t, ..., x_N, t_N + \Delta t) \quad \forall \Delta t$, то такой случайный процесс называется *стационарным* (в этом случае для $N = 1$ зависимость от времени вовсе отсутствует).

Частными характеристиками случайных процессов являются среднее, дисперсия, моменты высших порядков. Все эти величины могут зависеть от времени в общем случае, но для стационарного процесса моменты от времени не зависят. Иногда, говорят, что процесс называется стационарным в широком (слабом) смысле, если его среднее и дисперсия не зависят от времени. Важной характеристикой случайного процесса является автоковариационная функция или автокорелляционная функция, её нормированный вариант. Для стационарного процесса автоковариационная функция зависит от разности времён $|t_1 - t_2|$.

С точки зрения практической оценки моментов случайного процесса и его автоковариационной функции важно понятие *эргодичности*: когда усреднение по различным реализациям случайного процесса можно заменить на усреднение по времени в рамках одной доступной реализации.

6.1.1 Случайные последовательности

Рассмотрим индекс i нумерующий моменты времени на равномерной сетке, и приведём простые примеры случайных последовательностей. *Белый шум*:

$$p(x_1, ..., x_N) = \prod_{i=1}^{N} p(x_i), \tag{6.1}$$

необходимо задать вид функции одноточечной функции плотности вероятности $p(x)$, таким образом сразу окажется задана многоточечная функция плотности вероятности для любого N. В каком-то смысле, белый шум является предельным случаем и сводится к набору из N независимых одинаково распределённых величин.

Марковская случайная последовательности:

$$p(x_1, ..., x_N) = \prod_{i=2}^{N} p(x_i|x_{i-1})p(x_1). \tag{6.2}$$

Необходимо задать одноточечную функцию плотности вероятности $p(x)$, а так же двухточечную функцию условной плотности вероятности $p(x_i|x_{i-1})$, тогда наш случайный процесс (последовательность) будет полностью задан.

Случайная последовательность с нормальным распределением (гауссовская случайная последовательность):

$$p(x_1, ..., x_N) = \frac{1}{\sqrt{(2\pi)^N \det \Sigma}} \exp\left(-\frac{1}{2}(\mathbf{x} - \mu, \Sigma^{-1}(\mathbf{x} - \mu))\right) \tag{6.3}$$

нужно задать некоторые правила, как вычислять параметры распределения для любых индексов $\mu_i = \mu(i)$, $\Sigma_{ij} = \Sigma(i, j)$. Рассмотрим некоторые конкретные примеры случайных последовательностей с нормальным распределением.

6.2 Линейные модели авторегрессии скользящего среднего

6.2.1 Модель авторегрессии первого порядка

Пусть a_i — независимы и одинаково распределены $a_i \sim N(0, \sigma_0^2)$, а интересующий нас наблюдаемый случайный процесс x_i подчиняется разностному уравнению:

$$x_i = \phi x_{i-1} + a_i, \tag{6.4}$$

где ϕ некоторый параметр. $a_i = 0$, $x_i = 0$ для $i < 0$. Случайные величины a_i иногда называют генерирующим процессом. Разностное уравнение (6.4) задаёт *процесс авторегрессии*.

Видно, что $E[x_i] = 0$, значит $\mu_i = 0$. Для того, чтобы найти Σ_{ij}, решим разностное уравнение (6.4) методом производящих функций. Пусть $X(w) = \sum_{i=0}^{\infty} x_i w^i$, $A(w) = \sum_{i=0}^{\infty} a_i w^i$, тогда $X(w) = A(w)(1 - \phi w)^{-1}$. Раскладывая знаменатель в степенной ряд, и затем меняя порядок суммирования, получим, что

$$x_i = \sum_{m=0}^{\infty} \phi^m a_{i-m}, \tag{6.5}$$

откуда сразу видно, что каждый x_i распределён нормально, так как является суммой нормально-распределённых величин. Подсчитаем ковариацию

между x_i и x_j:

$$E\left[x_i x_j\right] = \sum_{n=0}^{\infty}\sum_{m=0}^{\infty}\phi^{n+m}E\left[a_{i-n}a_{j-m}\right] = \\ = \sigma_0^2\sum_{m=0}^{\infty}\phi^{2m+|i-j|} = \sigma_0^2\frac{\phi^{|i-j|}}{1-\phi^2} \quad (6.6)$$

Откуда сразу видно условие стационарности рассматриваемого случайного процесса $|\phi| < 1$.

Теперь можно сказать, что

$$\Sigma_{ij} = \sigma_0^2\Sigma_{ij}^{(0)}, \quad (6.7)$$

$$\Sigma_{ij}^{(0)} = \frac{\phi^{|i-j|}}{1-\phi^2}. \quad (6.8)$$

Предположим, что мы располагаем N последовательными измерениями из реализации такого случайного процесса, тогда можно расписать функцию логарифма правдоподобия:

$$\ln p(x_1,...,x_N|\phi,\sigma_0^2) = -\frac{N}{2}\ln\sigma_0^2 - \frac{1}{2}\ln\det\Sigma^{(0)}(\phi) - \frac{1}{2\sigma_0^2}(\mathbf{x},(\Sigma^{(0)}(\phi))^{-1}\mathbf{x}) \quad (6.9)$$

Параметр ϕ находится численно из уравнения

$$\operatorname{tr}\frac{\partial\Sigma_0}{\partial\phi}\left(\Sigma_0^{-1} - \frac{1}{\sigma_0^2}\Sigma_0^{-1}\mathbf{x}\otimes\mathbf{x}\Sigma_0^{-1}\right) = 0, \quad (6.10)$$

а параметр σ_0^2 выражается как

$$\sigma_0^2 = \frac{1}{N}(\mathbf{x},\Sigma_0^{-1}(\phi)\mathbf{x}), \quad (6.11)$$

Замечание, если пренебречь слагаемым $\ln\det\Sigma_0(\phi)$, то задача сводится к минимизации квадратичной формы:

$$\phi = \arg\min_{\phi}(\mathbf{x},\Sigma_0^{-1}(\phi)\mathbf{x}) \quad (6.12)$$

Кстати говоря, матрица Σ_0 по построению является симметричной тёплицевой матрицей, что подразумевает существование алгоритмов умножения на вектор за $O(N\log N)$ и обращения за $O(N^2)$.

6.2.2 Модели авторегрессии порядка p

Предыдущий подход можно обобщить:

$$x_i = \phi_1 x_{i-1} + ... + \phi_p x_{i-p} + a_i, \quad (6.13)$$

иногда для удобства вводят оператор сдвига по времени $Bx_i \equiv x_{i-1}$ и записывают разностное уравнение в операторном виде:

$$\Phi_p(B)x_i = a_i, \tag{6.14}$$

$$\Phi_p(B) \equiv 1 - \phi_1 B - ... - \phi_p B^p. \tag{6.15}$$

Процедура оценки параметров $\phi_1, ..., \phi_p$ аналогична предыдущему случаю. Можно показать, что

$$\Sigma_{ij} = \sigma_0^2 \left(A_1 w_1^{-|i-j|} + ... + A_p w_p^{-|i-j|} \right), \tag{6.16}$$

где $w_1, ..., w_p$ — корни так называемого характеристического уравнения:

$$1 - \phi_1 w - \phi_2 w^2 - ... - \phi_p w^p = 0. \tag{6.17}$$

Из формулы (6.16) видно, что процессы авторегрессии всегда имеют автоковариационную функцию в виде суммы затухающих экспонент и гармонических функций, в случае если решения характеристического уравнения состоят из комплексных чисел. Можно рассуждать и в обратную сторону: если выборочная автоковариационная функция некоторой реализации случайного процесса выглядит как сумма затухающих экспонент и гармонических функций, то разумно попытаться представить этот процесс моделью процесса авторегрессии.

6.2.3 Модель скользящего среднего

Рассмотрим другой вид разностных уравнений, сохраняя предыдущие предположения:

$$x_i = a_i - \theta_1 a_{i-1} - ... - \theta_q a_{i-q}, \tag{6.18}$$

или в операторном виде

$$x_i = \Theta_q(B)a_i, \tag{6.19}$$

$$\Theta_q(B) \equiv 1 - \theta_1 B - ... - \theta_q B^q. \tag{6.20}$$

Такой процесс называется *процессом скользящего среднего.* Действительно, в правой части уравнения записано выражение для взвешенной суммы. Сразу видно, что $\Sigma_{ij} = 0$, $\forall i, j : |i-j| > q$, а для $|i-j| \leq q$ соответствующие Σ_{ij} вычисляются как комбинация констант $\theta_1, ..., \theta_q$. Процедура оценки этих параметров $\theta_1, ..., \theta_q$ аналогична предыдущим случаям.

6.2.4 Модели авторегрессии интегрированного скользящего среднего

В самом общем виде рассматриваются модели авторегрессии интегрированного скользящего среднего, которые задаются уравнением

$$\Phi_p(B)(1-B)^d x_i = \Theta_q(B)a_i, \tag{6.21}$$

где $\Phi_p(B)$ и $\Theta_q(B)$ задаются уравнениями (6.15) и (6.20) соответственно.

Такая модель в записи часто обозначается как ARIMA(p, d, q). Как видно из уравнений (6.21), если мы зададим значения p, d, и q, т.е. определим количество необходимых нам параметров модели, то соответствующие $\phi_1, ..., \phi_p$, $\theta_1, ..., \theta_q$ и σ_0^2 определяются исходя из доступной реализации с помощью метода максимального правдоподобия.

Однако, процедура подбора p, d, и q выполняется перебором и называется *идентификацией модели*. Сравнение нескольких моделей описывающих одни и те же данные, но имеющие разный набор p, d, и q происходит по нескольким критерием: невязки модели не имеют остаточных корреляций, значение правдоподобия максимально, количество параметров минимально возможное (т.е. простота модели) при прочих равных условиях.

6.2.5 Прогнозирование

Прогноз — условная плотность вероятности (либо условное среднее) случайного процесса в некоторые моменты времени, когда значения не известны, при условии известных значений. Моменты времени могут быть либо в будущем, либо в прошлом, либо в другие моменты, когда не было измерений.

Пусть $x_1, ..., x_n$ известные значения случайного процесса в некоторые моменты времени, а $\hat{x}_{n+1}, ..., \hat{x}_{n+l}$ неизвестные значения в некоторые моменты времени. Для краткости обозначим из векторами $\mathbf{x}$ и $\hat{\mathbf{x}}$, соответственно. Тогда искомая условная многоточечная плотность вероятности:

$$p(\hat{x}_{n+1}, ..., \hat{x}_{n+l} | x_1, ..., x_n) = \frac{p(\hat{x}_{n+1}, ..., \hat{x}_{n+l}, x_1, ..., x_n)}{p(x_1, ..., x_n)} = \frac{p(\hat{\mathbf{x}}, \mathbf{x})}{p(\mathbf{x})}. \quad (6.22)$$

Числитель и знаменатель подчиняются многоточечной функции плотности вероятности исследуемого процесса, будем считать, что это нормальная плотность вероятности с нулевым средним, а параметры матрицы ковариации известны, например были найдены ранее с помощью метода максимального правдоподобия:

$$p(\hat{x}_{n+1}, ..., \hat{x}_{n+l} | x_1, ..., x_n) = \frac{\sqrt{(2\pi)^n \det \Sigma_{11}}}{\sqrt{(2\pi)^l \det \Sigma}} \frac{\exp\left(-\frac{1}{2}(\mathbf{x}', \Sigma^{-1}\mathbf{x}')\right)}{\exp\left(-\frac{1}{2}(\mathbf{x}, \Sigma_{11}^{-1}\mathbf{x})\right)}, \quad (6.23)$$

где блочный вектор

$$\mathbf{x}' = \begin{pmatrix} \mathbf{x} \\ \hat{\mathbf{x}} \end{pmatrix}, \quad (6.24)$$

а матрица Σ задаётся в блочном виде:

$$\Sigma = \begin{pmatrix} \Sigma_{11} & \Sigma_{12} \\ \Sigma_{21} & \Sigma_{22} \end{pmatrix}. \quad (6.25)$$

Используем формулу Фробениуса для обращения блочной матрицы, чтобы упростить выражение, тогда:

$$\Sigma^{-1} = \begin{pmatrix} \Sigma_{11}^{-1} + \Sigma_{11}^{-1}\Sigma_{12}H^{-1}\Sigma_{21}\Sigma_{11}^{-1} & -\Sigma_{11}^{-1}\Sigma_{12}H^{-1} \\ -H^{-1}\Sigma_{21}\Sigma_{11}^{-1} & H^{-1} \end{pmatrix}, \quad (6.26)$$

где

$$H \equiv \Sigma_{22} - \Sigma_{21}\Sigma_{11}^{-1}\Sigma_{12}. \tag{6.27}$$

Окончательно находим:

$$p(\hat{x}_{n+1}, ..., \hat{x}_{n+l}|x_1, ..., x_n) = \\ = \frac{1}{\sqrt{(2\pi)^l \det H}} \exp\left(-\frac{1}{2}(\hat{\mathbf{x}} - \Sigma_{21}\Sigma_{11}^{-1}\mathbf{x}, H^{-1}\left(\hat{\mathbf{x}} - \Sigma_{21}\Sigma_{11}^{-1}\mathbf{x}\right))\right), \tag{6.28}$$

откуда достаточно очевидно, что прогноз — тоже многомерное нормальное распределение, с известной матрицей ковариации H и некоторым условным средним $\Sigma_{21}\Sigma_{11}^{-1}\mathbf{x}$. Обратим внимание, что в этом случае точность прогноза (матрица ковариации H) не зависит от измеренных значений, а зависит только лишь от относительного расстояния во времени между моментами времени для прогноза и моментами времени известных значений.

Условное среднее зависит как от известных измерений, так и от корреляционных свойств процесса. Например, мы хотим предсказать среднее значение условного процесса, отстоящее в будущее на l шагов от последнего измеренного значения. В случае процесса скользящего среднего $\Sigma_{21} = 0$ в случае $l > q$, следовательно, условное среднее (прогноз) совпадает с безусловным средним, т.е. измерения уже не дают никакой подсказки о том, что может произойти через l шагов. В таких случаях говорят, что процесс имеет короткую память. Если мы имеем дело с процессом авторегрессии, то $\Sigma_{21} \neq 0$ даже для $l \gg p$, и тогда говорят, что процесс имеет длинную память.

6.3 Гауссовы процессы

В предыдущем разделе рассматривались линейные эмпирические модели для стохастических процессов для дискретного времени. Возникает естественное желание обобщить такие модели на случай непрерывного времени, или измерений, выполненных на неравномерной временной сетке. Например, при проведении наземных астрономических наблюдений, непредсказуемые погодные условия определяют возможность измерений в конкретный момент времени. Ранее известные измерения x_i и интересующие нас прогнозируемые величины $\hat{x}_l$ различались только индексом, нумерующим дискретные узлы сетки времени, а теперь для каждого измерения x_i должен быть дополнительно известен момент времени t_i, которому приписаны измерения.

Снова будем считать, что значение случайного процесса в каждый момент времени распределено согласно нормальному распределению с нулевым средним и некоторой дисперсией. Таким образом, многоточечная функция плотности вероятности может быть записана как

$$p(x_1, t_1; ...; x_N, t_N) = \frac{1}{\sqrt{(2\pi)^N \det \Sigma}} \exp\left(-\frac{1}{2}(\mathbf{x}, \Sigma^{-1}\mathbf{x})\right), \tag{6.29}$$

где $\mathbf{x}$ — вектор составленный из всех аргументов x_i, а элементы матрицы Σ зависят в общем случае от всех моментов времени $t_1, ..., t_N$.

Как и в прошлый раз, перед тем как применить метод максимального правдоподобия, нужно предложить некоторую параметрическую модель матрицы Σ. Для линейных моделей авторегрессии скользящего среднего параметрическая модель задавалась опосредованно, через рассмотрение разностных уравнений. Можно было бы поступить аналогично, рассматривая теорию стохастических дифференциальных уравнений Ито (см., например, Степанов 2012), однако на практике поступают более простым образом. Оказывается, можно доказать (см., например, Rasmussen и Williams 2006), что на роль Σ годится любая матрица заданная через так называемое *ядро* следующим образом:

$$(\Sigma)_{ij} = K(|t_i - t_j|), \tag{6.30}$$

где $K(\Delta t)$ — ядро, т.е. любая функция, которая удовлетворяет всем свойствам функции автоковариации стационарного случайного процесса. Например, $K(0) \geq |K(\Delta t)|$, $K(-\Delta t) = K(\Delta t)$, и т.п. Определение (6.30) позволяет гарантировать, что Σ удовлетворяет свойствам матрицы ковариации, таким как симметричность и положительная определённость.

Некоторые примеры ядер $K(\Delta t)$:

- Экспоненциальное ядро:[1]

$$K(\Delta t) = \sigma_0^2 \exp\left(\frac{-|\Delta t|}{l}\right), \tag{6.31}$$

  где σ_0^2 — параметр, определяющий дисперсию, а l — определяет характерный временной масштаб корреляции рассматриваемого процесса. Все параметры оцениваются исходя из метода максимального правдоподобия.

- Квадратично-экспоненциальное ядро:

$$K(\Delta t) = \sigma_0^2 \exp\left(\frac{-\Delta t^2}{2l^2}\right). \tag{6.32}$$

- Дробно-квадратичное ядро:[2]

$$K(\Delta t) = \sigma_0^2 \left(1 + \frac{\Delta t^2}{2\alpha l^2}\right)^{-\alpha}. \tag{6.33}$$

- Кроме того, оказывается, что если $K_1(\Delta t)$ и $K_2(\Delta t)$ — некоторые ядра, то их линейная комбинация тоже является ядром:

$$K(\Delta t) = \theta_1 K_1(\Delta t) + \theta_2 K_2(\Delta t), \tag{6.34}$$

  где параметры θ_1 и θ_2 оцениваются исходя из метода максимального правдоподобия.

[1] Ядро процесса Орнштейна-Уленбека, который является прямым аналогом процесса авторегрессии первого порядка для непрерывного времени.

[2] Rational quadratic

Последнее свойство открывает возможности для бесконечного комбинирования ядер межу собой. Если в случае линейных моделей авторегрессии скользящего среднего приходилось перебирать различные значения параметров p, d и q, то теперь количество потенциальных моделей значительно больше. Отметим, что для гауссовых процессов техническая трудность сопряжена с тем, что результирующая матрица Σ не имеет тёплицевой структуры, значит поиск неизвестных параметров требует большего количества вычислений.

После того как параметры ядра оценены с помощью метода максимального правдоподобия, прогнозирование осуществляется в соответствии с подходом из раздела 6.2.5. Моменты времени прогнозирования могут быть в прошлом, в будущем, либо в другие моменты времени, когда не проводились измерения. Последний случай, например, популярен как метод интерполяции измерений на равномерную сетку по времени для последующего применения методов машинного обучения для анализа набора различных реализаций случайных процессов.

6.4 Спектральная плотность мощности. Теорема Винера-Хинчина

Спектральная плотность мощности наряду с автоковариационной функцией $\gamma(\tau)$ случайного процесса является полезным инструментом анализа корреляционных свойств стационарных случайных процессов. Пусть $x(t)$ — действительный стационарный случайный процесс с нулевым средним. Рассмотрим следующую величину:

$$A_T(\omega) \equiv \frac{1}{\sqrt{2T}\sqrt{2\pi}} \int_{-T}^{T} dt x(t) \exp(-i\omega t). \tag{6.35}$$

Видно, что $\mathrm{E}\left[A_T\right] = 0$ для любого значения ω равно нулю, поэтому дисперсия величины $A_T(\omega)$ запишется следующим образом:

$$\begin{aligned}
\mathrm{E}\left[|A_T(\omega)|^2\right] &= \frac{1}{4\pi T} \int_{-T}^{T}\int_{-T}^{T} dt_1 dt_2 \exp(-i\omega(t_1 - t_2))\mathrm{E}\left[x(t_1)x(t_2)\right] = \\
&= \frac{1}{4\pi T} \int_{-T}^{T}\int_{-T}^{T} dt_1 dt_2 \exp(-i\omega(t_1 - t_2))\gamma(t_1 - t_2) = \\
&= \frac{1}{4\pi T} \int_{-2T}^{2T} d\tau \int_{-(T-|\tau|/2)}^{T-|\tau|/2} dt \exp(-i\omega\tau)\gamma(\tau) = \\
&= \frac{1}{2\pi} \int_{-2T}^{2T} d\tau \left(1 - \frac{|\tau|}{2T}\right) \exp(-i\omega(t_1 - t_2))\gamma(t_1 - t_2).
\end{aligned} \tag{6.36}$$

Следовательно,

$$S(\omega) \equiv \lim_{T\to\infty} \frac{\mathrm{E}\left[|\hat{x}_T(\omega)|^2\right]}{2\pi T} = \frac{1}{2\pi} \int_{-\infty}^{\infty} d\tau \gamma(\tau) \exp(-i\omega\tau), \tag{6.37}$$

где $S(\omega) \geq 0$ называется *спектральной плотностью мощности*, а $\hat{x}_T(\omega)$ — преобразование Фурье «урезанного» случайного процесса $x_T(t)$. Советский математик Александр Яковлевич Хинчин доказал, что если $S(\omega) \geq 0$, то справедливо и обратное утверждение:

$$\gamma(\tau) = \int_{-\infty}^{\infty} d\omega S(\omega) \exp(i\omega\tau), \tag{6.38}$$

причём $\gamma(\tau)$ — автоковариационная функция некоторого стационарного случайного процесса, т.е. для такой функции выполняются все необходимые свойства, которыми должна обладать автоковариационная функция. Полученное утверждение носит имя теоремы Винера-Хинчина и является одним из практичных способов вычисления выборочной автоковариацонной функции.

Речь идёт о том, чтобы применить к выборке отрезков случайного процесса преобразование Фурье, что особенно удобно делать при рассмотрении равномерной сетки времени, а затем вычислить выборочное среднее согласно определению (6.37). Далее полученная оценка спектральная плотности мощности может быть преобразована в автоковариационную функцию с помощью (6.38).

Аналогичное свойство можно доказать для эргодического случайного процесса. В этом случае, каждая реализация содержит в себе ансамбль реализаций, и вместо усреднения по ансамблю используется усреднение по времени:

$$\begin{aligned}
\gamma(\tau) &= \lim_{T\to\infty} \frac{1}{2T} \int_{-T}^{T} dt x_T(t) x_T(t+\tau) = \\
&= \frac{1}{8\pi^2} \lim_{T\to\infty} \frac{1}{T} \int_{-T}^{T} dt \int_{-\infty}^{\infty} d\omega \int_{-\infty}^{\infty} d\omega' \hat{x}_T(\omega) \hat{x}_T(\omega') \exp(it(\omega+\omega')) \exp(i\tau\omega') = \\
&= \int_{-\infty}^{\infty} d\omega \exp(i\tau\omega) \lim_{T\to\infty} \frac{|\hat{x}_T(\omega)|^2}{2\pi T}.
\end{aligned} \tag{6.39}$$

6.5 Согласованный фильтр

Согласованный фильтр[3] — один из простых но изящных способов решения задачи детектирования. Теперь про случайный процесс известно, что он составлен из суммы детерминированного сигнала $s(t)$ и некоторого аддитивного шума $\epsilon(t)$ с известными характеристиками:

$$x(t) = s(t) + \epsilon(t). \tag{6.40}$$

Предположим, что $\epsilon(t)$ — белый шум с нулевым средним и дисперсией σ_0^2. Кроме того, предположим, что сигнал $s(t)$ состоит из импульса известной

[3] matched filter

формы $s_0(\Delta t)$ и с конечным носителем $[0, T]$, который появляется или не появляется в некоторый момент времени t_0:

$$s(t) = s_0(t - t_0). \tag{6.41}$$

Задача детектирования состоит в том, чтобы понять был ли в сигнале импульс $s_0(\tau)$ или его не было, а был только аддитивный шум. Если амплитуда импульса много больше амплитуды шума, то работает наивный подход — детектирование по порогу сигнала, но при наличии шума достаточной мощности (дисперсии σ_0^2) результирующий (измеряемый, наблюдаемый) сигнал $x(t)$ может быть слабо различим для случаев присутствия сигнала и его отсутствия. Итак, нам бы хотелось получить ответ на вопрос передаётся ли в данный момент импульс или не передаются. Заметим, что успешное решение данной задачи может быть положено в основу простейшей линии цифровой передачи данных: например, импульс $s_0(\tau)$ будет обозначать логическую единицу, а импульс $-s_0(\tau)$ логический ноль.

Итак, рассмотрим линейный фильтр в общем виде:

$$x'(t) = \int_0^T d\tau h(\tau) x(t - \tau), \tag{6.42}$$

где $h(\tau)$ некоторая функция, задающая фильтр, которую иногда называют импульсной характеристикой. Такой фильтр может быть реализован как чисто аналоговым образом, так и цифровым.

Тогда сигнал преобразуются фильтром как

$$s'_0(\Delta t) = \int_0^T d\tau h(\tau) s_0(\Delta t - \tau), \tag{6.43}$$

а шум в свою очередь

$$\epsilon'(t) = \int_0^T d\tau h(\tau) \epsilon(t - \tau). \tag{6.44}$$

Сразу заметим, что среднее $\epsilon'(t)$ по прежнему равно нулю в силу линейности фильтра и попробуем оценить дисперсию $\epsilon'(t)$:

$$\mathrm{E}\left[\epsilon'^2\right] = \int_0^T d\tau \int_0^T d\tau' h(\tau) h(\tau') \mathrm{E}\left[\epsilon(t - \tau)\epsilon(t - \tau')\right] = \\ = \sigma_0^2 \int_0^T d\tau h^2(\tau), \tag{6.45}$$

здесь используется предположение, что $\epsilon(t)$ белый шум, но полученный результат можно без труда обобщить на случай любой другой известной автоковариацонной функции $\epsilon(t)$.

Для того, чтобы надёжно заметить полезный сигнал $s'_0(\Delta t)$ с помощью детектирования по порогу, амплитуда сигнала должна быть много больше, чем возможная амплитуда шума $\epsilon'(t)$. Очевидно, что чем больше будет

это отношение амплитуд (в радио-технике его часто называют *отношением сигнал-шум*), тем увереннее будет происходить детектирование, и тем меньше будет вероятность ошибиться: обнаружить сигнал там, где его не было, либо пропустить сигнал там, где он был. Формально это требование запишется как следующая вариационная задача:

$$h(\tau) = \arg\max_{h(\tau)} \frac{s_0'(T)^2}{\mathrm{E}\left[\epsilon'^2\right]} = \frac{\left(\int_0^T d\tau h(\tau) s_0(T-\tau)\right)^2}{\sigma_0^2 \int_0^T d\tau h^2(\tau)}. \tag{6.46}$$

Видно, что $h(\tau)$ необходимо отнормировать, так как умножение на константу не меняет отношения сигнал-шум. Выберем следующую нормировку:

$$\int_0^T d\tau h(\tau) s_0(T-\tau) = 1. \tag{6.47}$$

Итак, запишем теперь следующий функционал и найдём его минимум:

$$\Phi\left[h(\tau)\right] = \sigma_0^2 \int_0^T d\tau h^2(\tau) - \lambda \left(1 - \int_0^T d\tau h(\tau) s_0(T-\tau)\right), \tag{6.48}$$

тогда условие минимума запишется в следующем виде:

$$\int_0^T d\tau g(\tau) \left(2\sigma_0^2 h(\tau) + \lambda s_0(T-\tau)\right) = 0, \tag{6.49}$$

где $g(\tau)$ произвольная функция. Видно, что утверждение (6.49) выполняется для любых $g(\tau)$ только когда

$$h(\tau) = -\frac{\lambda}{2\sigma_0^2} s_0(T-\tau). \tag{6.50}$$

Константа λ может быть найдена из условия нормировки, но так как условие нормировки было выбрано достаточно произвольно, и не влияет на результат, то окончательно примем $h(\tau) = s_0(T-\tau)$.

Мы получили достаточно красивый результат: получается, что согласованный фильтр сопоставляет входящий сигнал с известным:

$$x'(t) = \int_0^T d\tau s_0(T-\tau) x(t-\tau), \tag{6.51}$$

а форма преобразованного сигнала определяется как

$$s_0'(\Delta t) = \int_0^T d\tau s_0(T-\tau) s_0(\Delta t - \tau). \tag{6.52}$$

Глава 7

Проверка статистических гипотез

Пусть известна выборка из N независимых реализаций некоторой случайной величины $x_1, ..., x_N$, для краткости обозначаемая вектором $\mathbf{x}$. Предположим, что мы хотим вынести некоторое суждение о распределении этой случайной величины. Например, убедиться, что случайная величина распределена нормально, удостовериться, что случайная величина обладает нулевым средним, или проверить, что две независимые случайные величины имеют одинаковое среднее. В простейших случаях, например, когда доступная выборка имеет очень большой размер, ответ на эти вопросы может оказаться очевидным, например, после построения гистограммы или квантиль-квантильного графика. Однако, в тех случаях, когда ответ не очевиден, мы хотели бы иметь формальный математический аппарат, позволяющий вынести аргументированное суждение о статистических свойствах.

Итак, *проверка статистических гипотез* заключается в том, что нужно сделать выбор между двумя утверждениями, называемыми обычно *нулевой гипотезой* H_0 и *альтернативной гипотезой* H_1. Например:

- Нулевая гипотеза H_0: $p(x_i) = \frac{1}{\sqrt{2\pi}} \exp\left(-\frac{x_i^2}{2}\right)$.
- Альтернативная гипотеза H_1: $p(x_i) = \frac{1}{4} \exp\left(-\frac{|x_i|}{2}\right)$.

Утверждения вида $p(x_i) = p_0(x_i)$ (следовательно $p(\mathbf{x}) = \prod_{i=1}^{N} p_0(x_i)$) часто называют *простой нулевой гипотезой*, а утверждения $p(x_i) = p_1(x_i)$ — *простой альтернативной*, смысл здесь в том, что параметры распределения считаются известными и не подлежат определению. В противном случае, гипотезу называют *сложной*. Например, если $p(x_i) = p'_0(x_i|\theta)$ известно с точностью до неизвестных параметров θ, которые подлежат оцениванию в рамках проверки статистической гипотезы.

С формальной точки зрения у нас в руках есть только выборка независимых реализаций случайной величины, записываемая как вектор $\mathbf{x} \in \mathrm{R}^N$.

И фактически, мы ищем способ разбить пространство R^N на две части. При попадании конкретной выборки, обозначаемой вектором $\mathbf{x}$ в одну из которых, называемую *критической областью* C, нулевая гипотеза отвергается, и, соответственно, при попадании в другую, нулевая гипотеза принимается. Кстати, в этом задача проверки статистических гипотез имеет некоторое сходство и параллели с задачей бинарной классификации из области машинного обучения.

С практической точки зрения проще всего оказывается ввести некоторую новую действительную функцию $f(\mathbf{x})$, которая затем сравнивается с некоторым пороговым значением K, и, например, если $f(\mathbf{x}) > K$, то нулевая гипотеза отклоняется, и наоборот. Функция $f(\mathbf{x})$ и порог K определяются некоторым оптимальным образом для каждой конкретной задачи, собственно, данная глава посвящена примерам выбора статистических критериев.

Основная сложность состоит в том, что чаще всего нулевая гипотеза допускает, что реализация $\mathbf{x}$ может принимать любые значения в R^N с ненулевой вероятностью, это значит, что даже при выполнении нулевой гипотезы вектор $\mathbf{x}$ может случайно оказаться в критической области, и нулевая гипотеза будет ошибочно отвергнута. Неравноправие в терминологии (нулевая и альтернативная гипотезы) во многом связано с понятием ошибок первого и второго рода:

- *Ошибкой первого рода* называется ситуация, когда гипотеза H_0 на самом деле верна, но была отклонена в рамках проверки.
- *Ошибкой второго рода* называется ситуация, когда гипотеза H_1 на самом деле верна, но была принята гипотеза H_0.

Понятно, что стоимость последствий ошибок первого и второго рода различна, и определяется исходя из внешних соображений. Например, допустим что анализируются данные некоторого медицинского исследования, и проверяются две гипотезы. Нулевая гипотеза H_0 состоит в том, что у пациента имеется некоторое очень опасное заболевание, а альтернативная H_1 — в том, что пациент здоров. Последствия ошибки первого рода будут состоять в том, что лечение пациента не будет начато своевременно, что приведёт к непоправимым последствиям для его здоровья, или даже летальному исходу. В то время как последствия ошибки второго рода будут состоять в том, что пациенту придётся потратить время, пройти ряд дополнительных исследований, возможно, поволноваться, но конечном счёте выяснится, что ему ничего не угрожает. В данном случае, очевидно, что последствия ошибки первого рода для данного человека будут иметь более высокую стоимость.

Итак, в рамках проверки статистических гипотез нам хотелось бы так задать критическую область C, чтобы вероятность ошибки первого рода не превосходила некоторого приемлемого для нас порога α, называемого *уровнем значимости*:

$$P\left\{\text{Ошибка первого рода}\right\} = \int_C p_0(\mathbf{x})d\mathbf{x} \leq \alpha. \tag{7.1}$$

Заметим, что существует вырожденный (и не интересный) случай, когда всегда принимается основная гипотеза, поэтому на практике мы фиксируем ошибку первого рода на её верхнем допустимом уровне α.

Вероятность ошибки второго рода должна быть сделана по возможности минимальной:

$$P\left\{\text{Ошибка второго рода}\right\} = \int_{\bar{C}} p_1(\mathbf{x})d\mathbf{x} = 1 - \int_{C} p_1(\mathbf{x})d\mathbf{x}. \tag{7.2}$$

Интеграл $\int_C p_1(\mathbf{x})d\mathbf{x}$ часто называют *мощностью критерия*.

На практике, как упоминалось, большинство критериев строится с помощью статистики $f(\mathbf{x})$ и порога K, в этом случае удаётся определить параметрическое семейство критических областей $C(K)$ и выбрать среди них критическую область для требуемого уровня значимости α. Иногда дополнительно удаётся доказать, что предложенный критерий наиболее мощный, т.е. имеет минимальную ошибку второго рода при заданном уровне значимости α.

7.1 Критерий Неймана-Пирсона

Рассмотрим отношение правдоподобий

$$\Lambda_{H_1,H_0} \equiv \frac{p_1(\mathbf{x})}{p_0(\mathbf{x})} \tag{7.3}$$

Гипотеза H_0 отвергается в том случае, когда $\Lambda_{H_1,H_0} \geq K$, некоторого порогового значения, которое по сути является функцией уровня значимости $K = K(\alpha)$. Лемма Неймана-Пирсона состоит в том, что данный критерий является наиболее мощным (минимальная вероятность ошибки второго рода), среди всех критериев с уровнем значимости (вероятностью ошибки первого рода) α.

Рассмотрим следующий пример из книги Кельберт и Сухов 2017. Пусть, для простоты рассмотрения, у нас есть одно измерение x_1, и мы хотим найти критерий отношения правдоподобий с уровнем значимости $\alpha = 0.05$. Тогда построим отношение правдоподобий:

$$\Lambda_{H_1,H_0} = \frac{\sqrt{2\pi}}{4}\frac{\exp\left(-\frac{|x_1|}{2}\right)}{\exp\left(-\frac{x_1^2}{2}\right)}. \tag{7.4}$$

Понятно, что $\Lambda_{H_1,H_0} \geq K(\alpha)$ эквивалентно утверждению $x_1^2 - |x_1| \geq K'(\alpha)$, таким образом критическая область $C(\alpha)$ в нашем случае задаётся двумя полупрямыми:

$$|x_1| \geq \frac{1}{2} + t(\alpha), \tag{7.5}$$

$$|x_1| \leq \frac{1}{2} - t(\alpha). \tag{7.6}$$

Рассчитаем вероятность ошибки первого рода, чтобы найти конкретное значение порога $t(\alpha)$.

Если $t \geq \frac{1}{2}$, тогда

$$\alpha = \int_C p_0(x_1)dx_1 = \\ = \frac{1}{\sqrt{2\pi}} \left(\int_{-\infty}^{-\frac{1}{2}-t} \exp\left(-\frac{x_1^2}{2}\right) dx_1 + \int_{\frac{1}{2}+t}^{\infty} \exp\left(-\frac{x_1^2}{2}\right) dx_1 \right) = \\ = 1 - \operatorname{erf}\left(\frac{1}{2} + t\right), \quad (7.7)$$

откуда следует, что $t = \operatorname{erf}^{-1}(1-\alpha) - \frac{1}{2}$, и для $\alpha = 0.05$ получается критическая область

$$|x_1| \geq 1.38\ldots . \quad (7.8)$$

Если $t \leq \frac{1}{2}$, тогда

$$\alpha = 1 - \operatorname{erf}\left(\frac{1}{2} + t\right) + \operatorname{erf}\left(\frac{1}{2} - t\right), \quad (7.9)$$

и это уравнение не имеет решений для выбранного $\alpha = 0.05$.

В данном случае $|x_1|$ играет роль статистики критерия, и нам остаётся сравнить эту величину с пороговым значением, специально найденным так, чтобы гарантировать выбранный уровень значимости $\alpha = 0.05$.

7.2 Критерий Стьюдента

Рассмотрим другую задачу. Пусть известно N независимых реализаций некоторой случайной величины $x_1, ..., x_N$. Гипотеза H_0 состоит в том, что случайная величина распределена согласно нормальному закону с средним μ_0, гипотеза H_1 состоит в том, что случайная величина распределена согласно нормальному закону с некоторым другим неизвестным средним $\mu \neq \mu_0$. Подразумевается, что величина μ_0 может предсказывается нам какой-то теорией, и мы хотели бы проверить или опровергнуть теорию на основе измерений.

Понятно, что почти невероятно, что для конечного числа N выборочное среднее

$$m \equiv \frac{1}{N} \sum_{i=1}^{N} x_i, \quad (7.10)$$

в точности совпадёт с величиной μ_0, поэтому нужен критерий допускающий отклонение выборочного среднего m от ожидаемого μ_0 в разумных пределах. Некоторое представление о характерном масштабе разброса случайной величины можно получить подсчитав выборочную дисперсию:

$$s^2 \equiv \frac{1}{N-1} \sum_{i=1}^{N} (x_i - m)^2, \quad (7.11)$$

тогда ошибка выборочного среднего m составит $\frac{s}{\sqrt{N}}$.

Рассмотрим величину

$$T \equiv \frac{m - \mu_0}{\frac{s}{\sqrt{N}}}, \tag{7.12}$$

которая имеет смысл отношения отклонения выборочного среднего от μ_0 к ошибке выборочного среднего, т.е. к корню дисперсии величины m. Иначе говоря, это относительное отклонение, заданное в единицах ошибки выборочной оценки среднего. Интуитивно понятно, что чем больше это относительно отклонение, тем менее вероятно равенство $m = \mu_0$.

Британский учёный Вильям Госсет (известный под псевдонимом Стьюдент) показал, что при выполнении H_0 такая величина T распределена в соответствии с распределением Стьюдента $p_{N-1}(T)$ с $N-1$ степенью свободы. Напомним, что плотность вероятности распределения Стьюдента задаётся следующим образом:

$$p_\nu(x) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi}\Gamma\left(\frac{\nu}{2}\right)} \left(1 + \frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}, \tag{7.13}$$

где целочисленный параметр ν называют числом степеней свободы распределения.

Значит можно вычислить вероятность ошибки первого рода для критерия $|T| > a$:

$$\alpha = \int_{-\infty}^{-a} p_{N-1}(T)dT + \int_{a}^{\infty} p_{N-1}(T)dT = 2\gamma_{N-1}(a), \tag{7.14}$$

где a — некоторый порог, который будет определён исходя из требуемого уровня значимости критерия. К несчастью, интеграл не выражается в элементарных функциях, поэтому просто обозначим кумулятивную функцию распределения Стьюдента как $\gamma_{N-1}(a)$, тогда решение уравнения записывается через обратную функцию, называемую квантильной функцией, как $a = t_{N-1}\left(\frac{\alpha}{2}\right)$. Итак, если $|T| \geq t_{N-1}\left(\frac{\alpha}{2}\right)$, то гипотеза H_0 отклоняется на уровне значимости α. Конкретное значение функции $t_{N-1}\left(\frac{\alpha}{2}\right)$ может быть найдено численно с помощью готовых программных пакетов, а до повсеместного распространения компьютерной техники издавались специальные таблицы значений этой функции.

Полученный результат часто формулируют в терминах доверительных интервалов (т.е. интервал, который накрывает настоящее значение μ с вероятностью $1 - \alpha$):

$$P\left\{m - \frac{s}{\sqrt{N}} t_{N-1}\left(\frac{\alpha}{2}\right) < \mu < m + \frac{s}{\sqrt{N}} t_{N-1}\left(\frac{\alpha}{2}\right)\right\} = 1 - \alpha. \tag{7.15}$$

Для проверки статистических гипотез часто вместо порога $t_{N-1}\left(\frac{\alpha}{2}\right)$ используется так называемое p-значение[1], определяемое как значение кумулятивной функции распределения от величины статистики, в случае критерия

[1] p-value

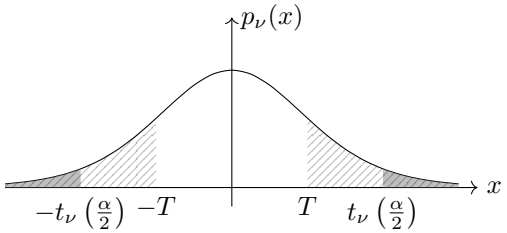

Рис. 7.1: Функция плотности вероятности распределения Стьюдента $p_\nu(x)$. Площадь обозначенная серым цветом равна уровню значимости критерия α. Площадь обозначенная диагональной штриховкой равна p-значению для показанного значения статистики T.

Стьюдента $p = \gamma_{N-1}(T)$. Благодаря монотонности кумулятивной функции любого распределения, критерий Стьюдента теперь записывается в виде $p(T) \geq \alpha$. Такая система единиц позволяет использовать единый формализм для многих статистических критериев, ведь величина p принимает значения на интервале $[0, 1]$, в отличии от порога, который имеет индивидуальные характерные значения для каждого критерия.

На рисунке 7.1 показана плотность вероятности распределения Стьюдента (7.13), пример значимости критерия α и соответствующий ему пороговый уровень, пример p-значения.

7.3 Критерий Пирсона

Пусть известно N независимых реализаций некоторой случайной величины $x_1, ..., x_N$. Нулевая гипотеза H_0 состоит в том, что случайная величина распределена с известным распределением $p_0(x)$.

Разобьём область значений x на K не пересекающихся интервалов, и для каждого интервала подсчитаем количество попаданий N_l в этот интервал. При нулевой гипотезе среднее число попаданий будет

$$e_l = Np_l = N \int_{D_l} p_0(x)dx, \tag{7.16}$$

и может быть рассчитано тем или иным образом для выбранного разбиения D_l и $p_0(x)$. При альтернативной гипотезе число попаданий будет произвольной величиной.

Рассмотрим величину

$$P_N \equiv \sum_{l=1}^{K} \frac{(N_l - e_l)^2}{e_l}. \tag{7.17}$$

Она имеет смысл взвешенной суммы квадратов отклонений числа попаданий в интервал от теоретических средних значений.

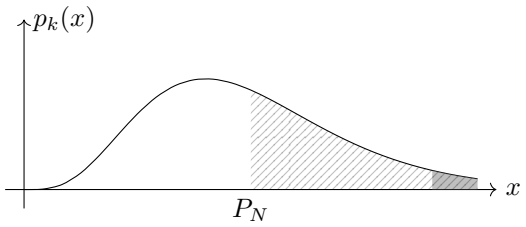

Рис. 7.2: Функция плотности вероятности распределения Пирсона $p_k(x)$. Площадь обозначенная серым цветом равна уровню значимости критерия α. Площадь обозначенная диагональной штриховкой равна p-значению для показанного значения статистики P_N.

Британский учёный Карл Пирсон доказал, что

$$\lim_{N\to\infty} P\left\{P_N > \alpha\right\} = \int_{\alpha}^{\infty} p_{N-1}(x)dx, \tag{7.18}$$

где плотность вероятности χ^2-распределения Пирсона с k степенями свободы задаётся выражением:

$$p_k(x) = \frac{1}{2^{k/2}\Gamma\left(\frac{k}{2}\right)} x^{k/2-1} \exp\left(-\frac{x}{2}\right), \tag{7.19}$$

где целочисленный параметр k называется числом степеней свободы.

Соответственно, гипотеза H_0 отклоняется с уровнем значимости α, если $P_N > \int_{\alpha}^{\infty} p_{N-1}(x)dx$.

На рисунке 7.2 показана плотность вероятности распределения Пирсона (7.19), пример значимости критерия α и соответствующий ему пороговый уровень, пример p-значения.

7.4 Критерий Колмогорова-Смирнова

Рассмотрим предыдущую задачу, но предложим другой способ её решения. Построим эмпирическую кумулятивную функцию распределения:

$$F_N(\mathbf{x}) \equiv \frac{1}{N}\sum_{i=1}^{N} I_{(-\infty,x]}(x_i), \tag{7.20}$$

где $I_{(-\infty,x]}$ обозначает индикаторную функцию, равную 1 для $x_i < x$ и нулю в противном случае.

Теоретическую кумулятивную функцию распределения при нулевой гипотезе мы знаем:

$$F(x) = \int_{-\infty}^{x} p_0(x)dx. \tag{7.21}$$

Рассматривается величина

$$D_N \equiv \sup_x |F(x) - F_N(x)|, \tag{7.22}$$

которая имеет смысл максимального отклонения между эмпирической и теоретической кумулятивными функциями. Заметим, что при $x = \pm\infty$ эти две функции всегда совпадают по построению, значит максимальное отклонение достигается где-то между этими точками.

Советский математик Андрей Николаевич Колмогоров изучил свойства распределения величины K, определяемой как:

$$K \equiv \sup_{t\in[0,1]} |B(t)|, \tag{7.23}$$

где $B(t)$ — Броуновский мост (про Броуновский мост см., например, Степанов 2012), случайный нестационарный процесс, для которого, в частности, $p(0, t = 0) = p(0, t = 1) = 1$. Полученное распределение называют распределением Колмогорова, оно показывает насколько сильно по амплитуде отклоняется Броуновский мост. Плотность вероятности и кумулятивная функция такого распределения задаётся в виде бесконечного ряда, и вычисляются с помощью численных методов.

Оказывается, что величина $\sqrt{N}D_N$ стремится к $\sup_x |B(F(x))|$ по распределению при $N \to \infty$. Таким образом, если $\sqrt{N}D_N > K(\alpha)$, то нулевая гипотеза отвергается, где $K(\alpha)$ находится из квантильной функции распределения Колмогорова для требуемого уровня значимости α. Асимптотически, мощность критерия стремится к единице, это значит, что он не совершает ошибок второго рода.

При попытке обобщить критерий Колмогорова-Смирнова на большие размерности случайных величин возникает интересное препятствие — ответ критерия становится не инвариантен относительно аффинных преобразований, т.е. простая линейная замена переменных влияет на результат. Поэтому, например, для проверки принадлежности выборки векторов к многомерному нормальному распределению существуют отдельные критерии, построенные исходя из принципа инвариантности относительно аффинных преобразований. Обзор различных критериев приведён, например, в работе Henze 2002.

Глава 8

Машинное обучение

Рассматриваемые в этой главе методы решающих деревьев (см. раздел 8.1), случайного леса (см. раздел 8.3.2), изолирующего леса (см. раздел 8.3.4), нейронных сетей (см. раздел 8.5) — это методы, которые в настоящее время традиционно объединяют под названием методов *машинного обучения*[1].

При этом часто говорят, что машинное обучение решает задачу об аппроксимации, которая понимается в широком смысле. Например, пусть есть некоторый набор данных, состоящий из N векторов $\mathbf{x}_i \in \mathrm{R}^n$, называемых векторами *признаков*[2], и соответствующие им N значений y_i. Подразумевается, что y_i получен из $\mathbf{x}_i$ с помощью некоторого соотношения. Конкретный вид этого соотношения чаще всего неизвестен, либо вычислительно затратен. Часто предполагается, что сама эта зависимость существует только в статистическом смысле, т.е. $\mathbf{x}_i$ и y_i могут быть отягощены некоторой неустранимой погрешностью. Мы бы хотели получить некоторую функцию $f(\mathbf{x})$, наилучшим образом аппроксимирующую неизвестную закономерность и одновременно с этим требующую разумного с практической точки зрения количества вычислений. Если y_i являются действительными числами, то говорят о *задаче регрессии*. Если y_i могут принимать дискретные значения, то говорят о *задаче классификации*, а сами y_i называют *метками*[3] классов.

Приведём другой пример. Пусть есть выборка из N реализаций случайного вектора $\mathbf{x}_i$, мы бы хотели получить такую функцию $f(\mathbf{x})$, которая аппроксимирует многомерную функцию плотности вероятности рассматриваемой случайной величины $p(\mathbf{x})$. Заметим, что в одномерном случае, когда $\mathbf{x}_i \in \mathrm{R}$, простой и эффективный подходом к решению этой задачи — построить гистограмму. В многомерном случае число требуемых бинов растёт экспоненциально с ростом размерности исследуемых данных n, понятно, что это создаёт как вычислительные сложности, так и делает гистограмму бесполезной, так как рано или поздно размер выборки N окажется много

[1] *machine learning*
[2] *features*
[3] *labels*

меньше числа бинов.

Или, например, пусть есть выборка из N реализаций случайного вектора $\mathbf{x}_i$, и пусть есть некоторая случайная величина $\mathbf{z}$ с нормальным распределением, нулевым средним и единичной дисперсией. Мы бы хотели получить такую функцию $f(\mathbf{z})$, чтобы случайный вектор $\xi = f(\mathbf{z})$ оказался бы распределён таким же образом как и случайный вектор $\mathbf{x}$. Всё это примеры задач аппроксимации для решения которых с тем или иным успехом применяются методы машинного обучения.

Заметим, что с задачей регрессии мы сталкивались в разделах 2 и 4.1.1. Задача оценки плотности вероятности по выборке реализаций случайного вектора решалась в главе 5. Во всех случаях сначала мы постулировали некоторый параметрический вид описываемой зависимости (например, линейная зависимость в разделе 2, или нормальное распределение в главе 5), а затем задача сводилась к оцениванию неизвестных параметров. Таким образом, из некоторого семейства функций выбиралась одна конкретная.

Понятно, что при работе с эмпирическими моделями чем шире класс функций из которых мы выбираем, тем лучше. Существует ряд классических методов, позволяющих параметризовать широкий класс функций, к таковым можно отнести ряды Фурье, всевозможные сплайны, интерполяционные полиномы, включая классические ортогональные полиномы, и т.п. Пожалуй, общее свойство всех этих методов состоит в том, что описываемое семейство функций допускает запись в виде формулы, пригодную для последующего практического использования. Машинное обучение представляет нам дополнительный способ алгоритмического задания аппроксимирующей функции. Нас удовлетворит, если для любого $\mathbf{x}$ можно будет вычислить соответствующее числовое значение $f(\mathbf{x})$ за разумное время.

Термин *обучение* используется, чтобы подчеркнуть, что искомый алгоритм не задаётся жёстко пользователем, а подбирается вычислительной машиной на основе известных данных. В простом случае, как например для метода нейронных сетей, это означает подбор неизвестных весовых коэффициентов. Для методов основанных на деревьях, мы подразумеваем процесс построения структур данных (деревьев), обладающих требуемыми свойствами.

Ещё раз заметим, что подобный алгоритм не просто должен уметь вычислять выходное значение, но и делать это в некотором смысле быстро. Рассмотрим, например, *метод k ближайших соседей* с помощью которого решают задачи классификации или регрессии. Идея метода крайне проста: если есть выборка из N векторов $\mathbf{x}_i$ и соответствующие им y_i, то предсказание для любой новой точки $\mathbf{x}$ может быть выполнено следующим образом. Найдём k точек исходной выборки ближайших к заданной точке $\mathbf{x}$ в эвклидовой метрике, и на основе соответствующих этим точкам метрик вычислим предсказание. Для задачи классификации потребуется определить к какому классу относится большинство ближайших точек, для задачи регрессии потребуется усреднить значения. Так или иначе, метод основан на свойстве k-мерных деревьев (см. раздел 1.4.3) эффективно находить для любой точки $\mathbf{x}$ ближайшие точки из заданного набора. Напомним, что асимптоти-

ческая вычислительная сложность такого поиска $O(\log(N))$.

Кроме вычислительных затрат, важной характеристикой моделей, построенных с помощью методов машинного обучения, является их качество, т.е. способность адекватно предсказывать требуемые величины. Количественные меры качества моделей называются *метриками качества* и рассматриваются далее в разделе 8.2.

Можно отметить два самых популярных применения методов машинного обучения в физических науках. Во-первых, построение эмпирических моделей, например, таких как калибровочные зависимости. Напомним, часто в физике бывает так, что построить строгую модель не удаётся — это оказывается либо сложно технически, либо не все влияющие факторы измеряются в эксперименте. Здесь в контексте машинного обучения и задачи классификации интересно упомянуть и о задаче фильтрации измерений.

Во-вторых, ускорение вычислений. Часто бывает так, что строгие физические модели достаточно затратны с вычислительной точки зрения, и, хотелось бы выполнить расчёт на некоторой сетке параметров, а затем интерполировать результаты для значений находящихся между узлов сетки. В этом случае методы машинного обучения могут быть хорошей альтернативой классическим методом интерполяции, особенно если речь идёт о пространствах большой размерности.

8.1 Решающее дерево

Решающее дерево[4], или *дерево принятия решений* — один из методов построения моделей машинного обучения для задач регрессии и классификации. Чаще всего решающее дерево используется как компонент для построения методов ансамблей, таких как, например, случайный лес (см. раздел 8.3.2).

В основе методов машинного обучения, основанных на деревьях, лежит идея о том, что кусочно-постоянная функция в многомерном пространстве может быть эффективно (с вычислительной точки зрения) представлена с помощью бинарного дерева. В одномерном случае, одним из наивных способов численного представления кусочно-постоянной функции является составление отсортированной таблицы пар x_i, y_i с последующим поиском x_i ближайшего слева к произвольному x. В многомерном случае такой подход не срабатывает из-за необходимости иметь число записей, зависящее от размерности пространства n экспоненциально.

На рисунке 8.1 показан пример решающего дерева для одномерного случая и соответствующая этому дереву кусочно-постоянная функция. Для того, чтобы вычислить значение этой функции для любого произвольного x необходимо выполнить поиск этого значения в соответствующем двоичном дереве. Такой поиск окажется неудачным[5] и завершится в листе, в котором

[4] *decision tree*

[5] Конечно, за исключением случаев, когда мы точно попали на границу двух интервалов. В этих точках, формально говоря, кусочно-постоянную функцию можно определить

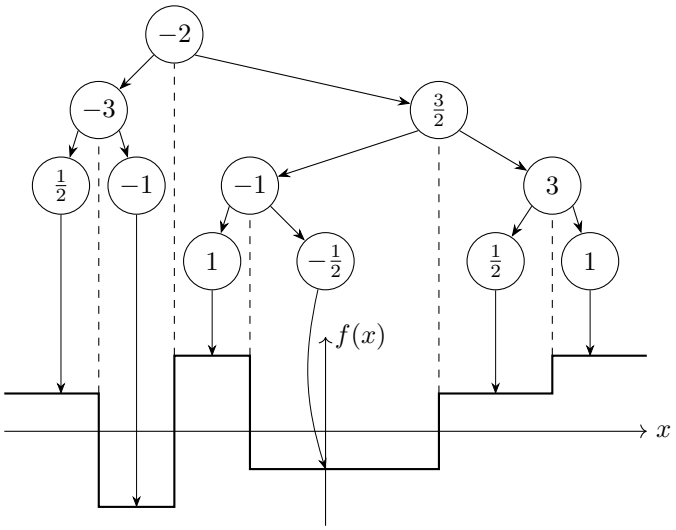

Рис. 8.1: Пример решающего дерева для одномерного случая и график соответствующей этому дереву кусочно-постоянной функции. Листья дерева хранят значения каждого интервала функции. Промежуточные узлы дерева хранят координаты границ интервалов.

записано значение функции на данном интервале. Иначе говоря, любому $x \in \mathrm{R}$ сопоставляется некоторый лист дерева с помощью операции поиска. При этом понятно, что все сопоставленные одному и тому же листу точки числовой прямой x образуют односвязную область — отрезок или луч.

Такой подход можно обобщить на пространства любой конечной размерности, вспомнив, что в разделе 1.4.3 обсуждалась организация коллекции из N многомерных векторов в k-мерное дерево, структуру данных, позволяющую эффективно искать элементы коллекции, напомним, что асимптотическая вычислительная сложность $O(\log N)$. Нетрудно представить, что аналогично одномерному случаю, для каждого $\mathbf{x} \in \mathrm{R}^k$ с помощью операции поиска (который, конечно же вновь окажется неудачным) можно сопоставить тот или иной лист дерева, в котором по прежнему записано значение функции для данного сегмента. Пример приведён на рисунке 8.2. Видно, что диаграмма неудачного поиска в k-мерном дереве представляет собой замощение многомерными параллелепипедами, причём это замощение не «пиксельное», состоящее из параллелепипедов одинакового размера, а напоминает скорее каменную кладку, когда прямоугольные камни разного размера тем не менее подогнаны без зазоров.

Итак, многомерное двоичное дерево может быть использовано для эффективного вычисления кусочно-постоянной функции. Причем значения этой функции определяются параметрами $\hat{y}_i$, заданными в листьях дере-

по разному.

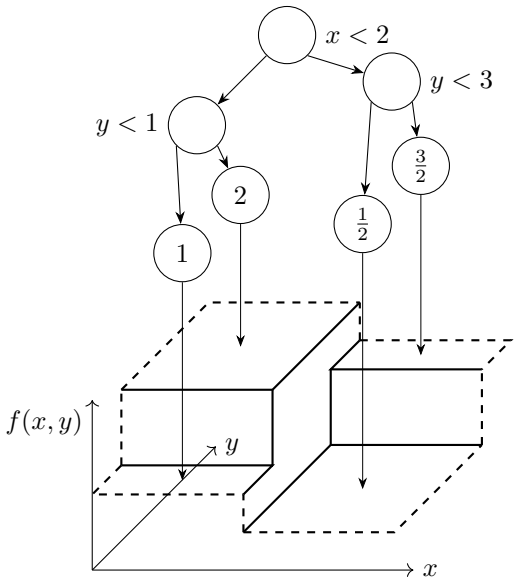

Рис. 8.2: Пример кусочно-постоянной функции двух координат. Штриховые линии показывают, что в данном направлении сегмент простирается за пределы графика. Сверху показано k-мерное дерево, соответствующее данной функции.

ва, а границы сегментов определяются значениями $\hat{x}_i$, k_i, заданными в узлах дерева, где k_i обозначает индекс координаты по которому производится разбиение. Обратим внимание, что в отличии от задачи хранения и поиска заданной коллекции многомерных векторов, величины $\hat{y}_i$ в каждом листе и $\hat{x}_i$, k_i в каждом промежуточном узле являются параметрами. Кроме того, параметром является число листьев дерева, так как каждый отдельный сегмент кусочно-постоянной функции представляется отдельным листом.

Возникает вопрос, как эффективно с точки зрения вычислительной сложности и оптимально с точки зрения аппроксимации некоторой функции подобрать неизвестные параметры $\hat{y}_i$, $\hat{x}_i$, и k_i, чтобы полностью задать дерево, которое бы представляло некоторую искомую кусочно-постоянную функцию? Понятно, что в разделе 1.4.3 мы стремились построить дерево, имеющее по возможности наименьшую глубину, что положительно сказывалось на эффективности поиска по коллекции. Теперь мы смещаем внимание на эффективность аппроксимации неизвестной зависимости кусочно-постоянной функцией, численно представляемой многомерным бинарным деревом.

При этом есть основания полагать, что вычислительная сложность сильно не пострадает. Во-первых, напомним, что даже случайное бинарное дерево имеет среднюю глубину пропорциональную логарифму числа узлов $O(\log N)$ (см. раздел 1.4.3). Во-вторых, частой практикой является ограничение глубины дерева, задаваемое перед построением дерева. В терминах кусочно-постоянной функции это эквивалентно ограничению числа индивидуальных сегментов функции.

8.1.1 Задача классификации

Универсального оптимального способа построения аппроксимирующего дерева не существует, как это и принято в машинном обучении. Предложено множество подходов, большинство которых имеют скорее интуитивное объяснение, чем строгое обоснование (см., например, L. Breiman и др. 1984), так как конечный критерием успешного решения задачи в машинном обучении — *метрики качества* (будут обсуждаться в разделе 8.2), которые показывают насколько эффективно метод работает с конкретными данными. На практике метрики качества всегда рассчитываются эмпирически и не могут быть предсказаны заранее. Этим объясняется общепринятая стратегия применения методов машинного обучения: решить одну и ту же задачу разными способами и выбрать лучший.

Тем не менее, различные подходы к построению деревьев имеют общие черты, поэтому для демонстрации рассмотрим один из простых и популярных способов построения решающего дерева для задачи классификации. Напомним, что задан набор из N векторов признаков $\mathbf{x}_i \in \mathrm{R}^n$, и соответствующие им N значений $y_i \in \{0, 1\}$, а задача состоит в том, чтобы предоставить функцию, которая могла бы быть вычислена для любого нового вектора признаков $\mathbf{x}$. Во-первых, понятно, что процедура построения дерева должна быть эффективна с вычислительной точки зрения, и для

этого было бы удобно использовать рекурсивное разделение, т.е. задать и рекуррентно применять одно универсальное правило разбиения данных на два подмножества, которые будут образовывать левое и правое поддеревья. Тогда асимптотическая вычислительная сложность в среднем составит $O(N \log N)$.

Во-вторых, одновременно с этим задача аппроксимации должна решаться эффективно с точки зрения точности. Поэтому используют *жадные*[6] способы построения дерева. Напомним, что жадным алгоритмом называется алгоритм, который принимает последовательность локально-оптимальных решений с целью получить решение, близкое к глобально-оптимальному. В нашем случае, строго гарантировать глобальную оптимальность не удаётся, но оказывается, что на практике следующий метод работает эффективно.

Если представить, что дерево единичной глубины, т.е. состоящее из одного корня и двух листьев, тогда параметры корня дерева $\hat{x}_0, k_0$ определят границу раздела пространства на две части, в каждой из которых функция заданная деревом будет принимать некоторое постоянное значение. С точки зрения здравого смысла понятно, что нам бы хотелось провести границу так, чтобы по обе стороны от неё содержащиеся там точки в большинстве своём принадлежали какому-то одному классу. В таком случае, количество ошибок неправильной классификации, совершаемых при использовании аппроксимации, было бы минимальным.

Одной из возможных количественных мер оценки чистоты выборки является *примесь Джини*[7]:

$$G \equiv p_0(1-p_0) + p_1(1-p_1) = 2p_0(1-p_0), \tag{8.1}$$

где введены обозначения

$$p_0 \equiv \frac{N_0}{N_0+N_1}, \tag{8.2}$$

$$p_1 \equiv \frac{N_1}{N_0+N_1}. \tag{8.3}$$

Здесь N_0 и N_1 обозначают количество точек принадлежащих нулевому и первому классам соответственно. Видно, что $G = 0$ либо если все точки принадлежат нулевому классу и тогда второй множитель равен нулю, либо если все точки принадлежат первому классу и тогда первый множитель равен нулю. Критерий легко обобщается на случай многомерной классификации:

$$G \equiv \sum_{j=0}^{K-1} p_j(1-p_j), \tag{8.4}$$

$$p_j \equiv \frac{N_j}{\sum_{k=0}^{K-1} N_k}. \tag{8.5}$$

[6] *greedy*
[7] *Giny impurity*

где K — полное число различных классов.

Примесь Джини имеет смысл аналогичный дисперсии для непрерывных величин. Это легко уяснить из следующего вспомогательного примера. Пусть задан случайный вектор β, реализации которого могут принимать лишь два значения: $(1,0)$ с вероятностью p_0 и $(0,1)$ с вероятностью $p_1 = 1 - p_0$. С помощью нехитрых вычислений можно убедиться, что среднее такого случайного вектора — (p_0, p_1), а матрица ковариации:

$$\operatorname{cov}\beta = \begin{pmatrix} p_0(1-p_0) & -p_0p_1 \\ -p_1p_0 & p_1(1-p_1) \end{pmatrix}, \tag{8.6}$$

откуда видно, что примесь Джини фактические является следом этой матрицы.

Идея поиска оптимального разделения состоит в том, чтобы подсчитать взвешенную сумму примесей Джини G для обоих частей (подвыборок) разбиения пространства признаков, а затем подобрать параметры таким образом, чтобы рассматривая сумма была минимально возможной:

$$\theta = \arg\min_{\theta} \left(\frac{N_l}{N_l + N_r} G_l + \frac{N_r}{N_l + N_r} G_r \right), \tag{8.7}$$

где N_l обозначает количество данных в полупространстве соответствующем левому поддереву, N_r — правому поддереву, G_l — примесь Джини для левого поддерева, G_r — для правого, $\theta \equiv (\hat{x}_0, k_0)$ параметры разбиения. Понятно, что все величины зависят от параметров разбиения θ, но для краткости на записи это не обозначено. Видно, что целевая функция равна нулю в двух случаях: когда в левом поддереве все элементы принадлежат нулевому классу, а в правом поддереве — первому, и наоборот. Оба этих случая нас устраивают с точки зрения эффективности аппроксимации.

Получив одно разделение далее индивидуально применим этот же принцип к левому и правому поддереву. Окончательно, каждый лист будет накрывать область пространства признаков, содержащую хотя бы одну точку данных, либо, если при построении задано ограничение по глубине дерева, несколько точек. Окончательно, каждому листу $\hat{y}_i$ припишем класс, который, например, наиболее часто встречается в этой области.

Теперь параметры всех промежуточных узлов дерева, т.е. параметры разбиения, и значения узлов дерева полностью заданы. Таким образом оказалось задано вполне конкретное дерево, задающее кусочно-постоянную функцию, которая, как мы считаем, адекватно аппроксимирует интересующую нас зависимость.

8.1.2 Задача регрессии

Если мы решаем задачу регрессии, тогда подход построения решающего дерева нужно изменить. Очевидно хотя бы то, что нужна другая функция для поиска оптимального разделения, так как примесь Джини (8.7) не может быть задана для y_i принимающих значения из непрерывного диапазона.

Один из наиболее естественных вариантов состоит в том, чтобы подсчитать смещённую выборочную дисперсию каждой подвыборки:

$$V \equiv \frac{1}{N} \sum_{i=1}^{N} (y_i - \bar{y})^2, \tag{8.8}$$

$$\bar{y} \equiv \frac{1}{N} \sum_{i=1}^{N} y_i. \tag{8.9}$$

Можно дать альтернативную интерпретацию формуле (8.8): если считать, что $\bar{y}$ — некоторое предсказание модели, как это обычно и бывает, то величина V задаёт среднеквадратичное отклонение данных от такого предсказания.

Чем меньше выборочная дисперсия, тем ближе друг к другу значения y_i в подвыборке. В случае нулевой дисперсии все значения равны друг другу, значит такое разбиение было бы для нас идеальным. Следовательно, можно определить критерий выбора параметров разбиения следующим образом:

$$\theta = \arg\min_{\theta} \left(\frac{N_l}{N_l + N_r} V_l + \frac{N_r}{N_l + N_r} V_r \right) = \\ = \arg\max_{\theta} \frac{N_l N_r}{(N_l + N_r)^2} (\bar{y}_l - \bar{y}_r)^2, \tag{8.10}$$

где как и ранее N_l — количество данных в полупространстве соответствующем левому поддереву, N_r — правому поддереву, V_l, V_r — дисперсии для левого и правого поддеревьев, соответственно, y_l, y_r — выборочные средние для левого и правого поддеревьев. На рисунке 8.3 приведён пример использования критерия (8.10) для разбиения данных.

На практике оказывается более удобно воспользоваться эквивалентной задачей максимизации квадрата разности выборочных средних (8.10), так как это не требует вычисления вторых моментов выборки. Равенство в выражении (8.10) можно доказать строго формально с помощью нехитрых, но громоздких вычислений.

Интересно, что в области компьютерного зрения критерий (8.10) был предложен японским исследователем Нобуюки Оцу (см. Otsu 1979), метод Оцу позволяет динамически подбирать порог для выделения ярких пикселей на тёмном фоне, либо границ объектов, если предварительно к изображению применён градиентный фильтр или фильтр Лапласа. В отличии, например, от модели смеси двух нормальных распределений (см. раздел 5.1), который тоже можно было бы применить для определения оптимального порога, в этом методе не делается предположений о распределении значений тёмных (фоновых) пикселей, и распределении значений светлых пикселей (пикселей объекта).

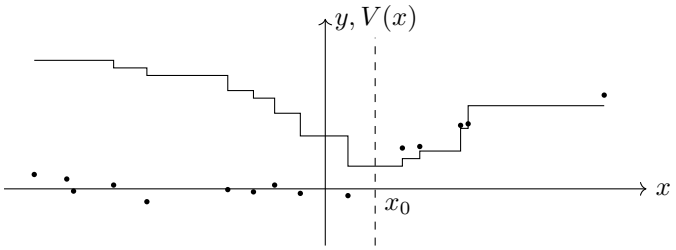

Рис. 8.3: Пример использования критерия (8.10) для разбиения данных. Точками обозначены записи некоторого набора данных, состоящего из пар x_i, y_i. Сплошная кривая показывает график функции $V(x_0)$, взвешенной суммарной дисперсии в двух подвыборках, разделённых значением x_0. Оптимальное значение $\hat{x}_0$ показано вертикальной штриховой линией.

8.2 Метрики качества и гиперпараметры

8.2.1 Метрики качества

В предыдущем разделе было приведено два простых примера методов машинного обучения для решения задач классификации и регрессии с помощью решающих деревьев. В последующих разделах приводится ещё больше примеров, таких, например, как методы на основе ансамблей деревьев (см. раздел 8.3), логистическая регрессия (см. раздел 8.4) и нейронные сети (см. раздел 8.5). Возникает важный и очевидный вопрос о том, как можно было бы характеризовать эффективность того или иного метода. Во-первых, как это иногда бывает с методами машинного обучения, сами алгоритмы построены таким образом, что строго не очевидно, почему тот или иной метод вовсе приводит к успеху. В отличии, например, от метода наименьших квадратов (глава 2) или метода максимального правдоподобия (глава 5), где вначале постулируется базовый принцип, а затем все последующие действия выводятся из него формально и с достаточной математической строгостью, в случае с машинным обучением алгоритмы часто строятся по принципу проб и ошибок, а в качестве обоснования приводятся интуитивные рассуждения. Например, в предыдущем разделе так и произошло — задача поиска оптимального решающего дерева была заменена на неэквивалентную последовательность отдельных задач оптимизации.

Во-вторых, многообразие доступных методов, очевидно, предполагает необходимость выбора некоторого наилучшего способа для решения задачи. Значит, необходимо иметь способ однородно охарактеризовать различные методы, применённые к решению одной и той же задачи.

В-третьих, чаще всего методы машинного обучения имеют некоторые настроечные параметры, которые не определяются непосредственно из данных, такие параметры принято называть *гиперпараметрами*. Например, в зависимости от выбранного метода гиперпараметрами могут быть предельная глубина решающего дерева (см. раздел 8.1); количество деревьев в ансамбле, или доля данных, выбранных для бутстрапа, (см. раздел 8.3); архитектура нейронной сети (см. раздел 8.5); и т.п. Это означает, что даже в рамках одного метода необходимо сравнение случаев с разным выбором гиперпараметров.

Сначала напомним, что алгоритмы машинного обучения могут быть охарактеризованы с точки зрения скорости работы: как в терминах асимптотической вычислительной сложности (см. раздел 1.2), так и с точки зрения измерения реального времени работы в конкретных условиях. Зачастую, время работы имеет немаловажное практическое значение, но верно и обратное — алгоритм, который быстро, но неправильно решает задачу, как правило представляет мало интереса.

Чтобы охарактеризовать насколько хорошо тот или иной метод решает исходную задачу используются *метрики качества*. Метрики качества рассматривают модель, построенную с помощью методов машинного обучения, как инструмент, способный давать предсказания, при этом не вникая

во внутреннее устройство модели, что и позволяет сравнивать между собой различные подходы. Например, для задачи регрессии в качестве метрики качества можно было бы выбрать среднеквадратичное отклонение[8] между предсказанием модели и настоящими измерениями:

$$\mathrm{MSE} \equiv \frac{1}{N} \sum_{i=1}^{N} \left(f(\mathbf{x}_i) - y_i \right)^2 , \tag{8.11}$$

где $\mathbf{x}_i$ — известные признаки, а y_i — заранее известный правильный ответ.

Такой выбор метрики качества для задачи регрессии не единственно возможный, в качестве альтернативы можно упомянуть, например, среднее абсолютное отклонение[9]:

$$\mathrm{MAE} \equiv \frac{1}{N} \sum_{i=1}^{N} \left| f(\mathbf{x}_i) - y_i \right| . \tag{8.12}$$

Конкретная метрика качества выбирается владельцем задачи исходя из представлений о том, какая именно проблема из реального мира решается в данном случае с помощью методов машинного обучения.

Метрики качества для задач классификации связаны с понятием *матрицы ошибок*[10], для случая бинарной классификации она выглядит следующим образом:

$$C \equiv \begin{bmatrix} \mathrm{TN} & \mathrm{FP} \\ \mathrm{FN} & \mathrm{TP} \end{bmatrix} , \tag{8.13}$$

и в каждой ячейке записано число событий, при использовании модели с набором данных с известными ответами:

- TN — число *истинно отрицательных*[11] событий, т.е. количество раз, когда модель назвала отрицательное событие отрицательным;
- FP — число *ложно положительных*[12] событий, т.е. количество раз, когда модель назвала отрицательное событие положительным;
- FN — число *ложно отрицательных*[13] событий, т.е. количество раз, когда модель назвала положительное событие отрицательным;
- TP — число *истинно положительных*[14] событий, т.е. количество раз, когда модель назвала положительное событие положительным.

[8] *mean squared error (MSE)*
[9] *mean absolute error (MAE)*
[10] *confustion matrix*
[11] *true negative*
[12] *false positive*
[13] *false negative*
[14] *true positive*

Понятно, что сумма TN, FP, FN и TP равна полному числу данных в исследуемом наборе, потому-что кроме рассмотренных четырёх вариантов других случаев быть не может. Видно, что колонки матрицы ошибок C соответствуют предсказанному классу, а строки соответствуют реальному классу. В случае, когда решается задача многоклассовой классификации, соответствующая матрица ошибок будет иметь столько же строк и колонок, сколько рассматриваемых классов. Матрица ошибок C не симметричная, потому-что в общем случае число ложно положительных и ложно отрицательных событий не обязаны совпадать, аналогично понятиям ошибок первого и второго рода из главы 7.

Подсчитав матрицу ошибок, можно ввести несколько метрик качества моделей для задач классификации, например, таких как *точность* (*accuracy*)[15] — доля правильно определённых классов:

$$\text{Accuracy} \equiv \frac{\text{TN} + \text{TP}}{N}. \tag{8.14}$$

Точность (*precision*) — доля правильно определённых положительных событий среди всех событий, названных положительными:

$$\text{Precision} \equiv \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{8.15}$$

при этом некоторое неопределённое число положительных событий может быть отклонено классификатором. *Полнота*[16] — доля найденных положительных событий по отношению к полному числу положительных событий:

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{8.16}$$

при этом некоторое число отрицательных событий может быть принято классификатором за положительные. Понятно, что используя различные комбинации значений из матрицы ошибок можно построить и другие метрики качества.

Выбор конкретной метрики качества, как говорилось ранее, осуществляется владельцем задачи в зависимости от решаемой задачи. Например, в некотором эксперименте могут измеряться величины $\mathbf{x}_i$ от смеси двух распределений (как в разделе 5) и мы используем некоторый классификатор для отделения событий, принадлежащих каждому из распределений. Допустим, один класс событий — шумовые события (отрицательные), а другой класс — интересующие нас измерения характеристик частиц космического излучения (положительные). В одном случае, нас могут интересовать статистические характеристики измеряемых величин, например спектр энергии,

[15] К сожалению, переводы многих терминов машинного обучения на русский язык в настоящее время всё ещё не устоялись. Так, в частности, под словом точность понимают две различных величины: *accuracy* и *precision*. Для устранения путаницы существуют сомнительные с точки зрения русского языка попытки перевода *accuracy* как *аккуратность*.

[16] *recall*

тогда нам хотелось бы, чтобы выборка положительных событий была бы как можно меньше загрязнена неправильно классифицированными отрицательными событиями, т.е. иметь высокую точность (precision), даже ценой того, что часть положительных событий окажется отсеяна. В другом случае, например, когда исследуемые события очень редки, может оказаться так, что нам не хотелось бы пропускать положительные события, т.е. иметь высокую полноту, пусть и ценой того, что в результирующей выборке окажется множество ложно положительных событий, которые, например, потребуют потратить время на дополнительную ручную проверку и анализ.

Кривая рабочей характеристики приёмника

Вообще говоря, в большинстве случаев классификатор основан на сравнении некоторой непрерывной величины t с заданным порогом T. Аналогично тому, как устроены статистические критерии из главы 7. В роли такой величины выступает, например, доля ближайших точек, принадлежащих положительному классу в методе k ближайших соседей (см. введение главы 8); доля точек, принадлежащих положительному классу в данном листе решающего дерева, ограниченного по глубине (см. раздел 8.1); доля деревьев, относящих точку к положительному классу в методе случайного леса (см. раздел 8.3.2); или оценка условной плотности вероятности принадлежности к положительному классу в методе логистической регрессии (см. раздел 8.4). Понятно, что настраивая такой порог T для одного и того же классификатора можно получить различные показатели, например, точности и полноты. Достаточно очевидно, что при улучшении точности с помощью изменения порога T полнота будет ухудшаться, но «курс обмена» будет различный для различных классификаторов.

Чтобы показать это более формально, обозначим условные плотности вероятность величины t для события отрицательного класса и положительного класса как $p(t|\{z=0\})$ и $p(t|\{z=1\})$ соответственно, а условные плотности вероятности для вектора признаков $\mathbf{x}$ обозначим как $p(\mathbf{x}|\{z=0\})$ и $p(\mathbf{x}|\{z=1\})$. На практике ни одна из этих функций заранее точно не известна, а классификатор строит некоторое преобразование $t=f(\mathbf{x})$, формально связывая соответствующие $p(t|z)$ и $p(\mathbf{x}|z)$. Понятно, что если для какой-то области пространства признаков соответствующие $p(\mathbf{x}|\{z=0\}) \approx p(\mathbf{x}|\{z=1\})$, то требовать от классификатора идеальной эффективности невозможно, потому-что исходные данные не предусматривают точного разделения.

Теперь мы можем записать теоретическое выражение для ожидаемой полноты, т.е. *доли истинно положительных событий*[17], в зависимости от заданного порога T:

$$\mathrm{TPR} = \int_T^\infty p(t|\{z=1\})dt, \tag{8.17}$$

[17] *true positive rate (TPR)*

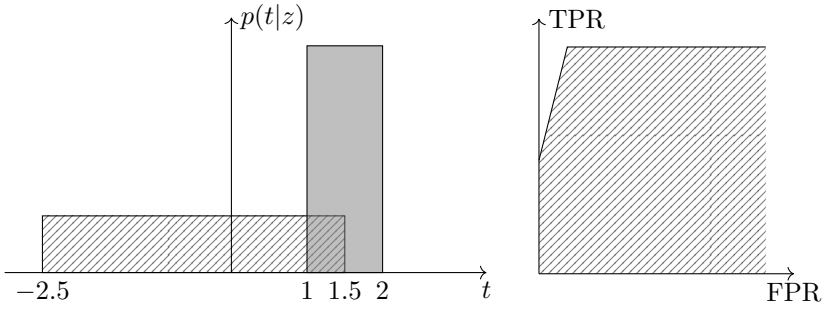

Рис. 8.4: На левом рисунке приведён пример графиков функций $p(t|\{z = 0\})$ (кривая со штриховкой под ней) и $p(t|\{z = 1\})$ (кривая с заливкой серым цветом под ней) для некоторого гипотетического классификатора. На правом рисунке показан соответствующий график рабочей характеристики приёмника. Штриховкой обозначена площадь под кривой рабочей характеристики приёмника (ROC AUC).

и *долю ложно положительных событий*[18]:

$$\text{FPR} = \int_T^\infty p(t|\{z = 0\})dt. \tag{8.18}$$

Уравнения (8.17) и (8.18) задают параметрическую кривую на графике зависимости $\text{TPR}(T)$ от $\text{FPR}(T)$. Исторически сложилось, что эта кривая называется *рабочей характеристикой приёмника*[19]. Кривая во-первых, показывает соотношение между полнотой и долей ложно положительных событий. Во-вторых, позволяет ввести ещё одну популярную метрику качества классификатора, известную как *площадь под кривой ROC*[20]. Эта метрика качества, очевидно, не зависит от настроечного параметра порога T и показывает насколько классификатор хорошо разделяет данные в принципе.

На рисунке 8.4 приводится пример графиков функций $p(t|\{z = 0\})$ и $p(t|\{z = 1\})$, а также соответствующий им график рабочей характеристики приёмника. Видно, что в данном примере наиболее поучительный интервал порога T — от 1 до 1.5. В этом случае, чтобы избавиться от $\frac{1}{8}$ доли шумовых событий приходится пожертвовать половиной всех положительных событий. При этом фиксация порога T меньше 1 или больше 1.5 не имеет практического смысла, так как приводит только лишь к ухудшению одной из метрик без улучшения другой метрики взамен. Соответствующий график рабочей характеристикой приёмника (ROC) состоит из трёх отрезков прямых линий: вертикальный отрезок соответствует порогу $T \geq 1.5$, наклонный отрезок — $T \in [1, 1.5]$ и горизонтальный отрезок — $T \leq 1$.

[18] *false positive rate (FPR)*
[19] *receiver operating characteristic (ROC)*
[20] *area under the curve (AUC)*

8.2.2 Подбор гиперпараметров и переобучение

Совершенно очевидно, что конкретная величина метрики качества зависит не только от исследуемой модели, но и от используемой для вычисления этой метрики выборки данных. Напомним, что, независимо от решаемой задачи, наши признаки имеют некоторое распределение $p(\mathbf{x})$, а соответствующие метки — условное распределение $p(y|\mathbf{x})$. В рамках построения моделей нам доступна некоторая выборка из совместного распределения $\mathbf{x}_i$, y_i, которая обычно называется *учебной выборкой*[21] и используется для построения модели (или, как говорят иначе, её обучения), т.е., например, для определения оптимальных разбиений при построении решающего дерева (см. раздел 8.1) или определения весов некоторой нейронной сети (см. раздел 8.5).

Однако, эффективность модели мы хотим охарактеризовать для всех возможных данных, а не для конкретной подвыборки, иначе такая эмпирическая модель просто не может использоваться для предсказаний. Модель и нужна для того, чтобы достоверно предсказывать результат при новых условиях $\mathbf{x}$, для которых соответствующая метка y неизвестна. Иначе говоря, в идеальном мире мы должны были бы рассчитывать метрики типа (8.11), (8.12), или (8.13) не по выборке, а по настоящему распределению, т.е., например:

$$\mathrm{MSE} = \int d\mathbf{x}dy p(y|\mathbf{x})p(\mathbf{x})\left(f(\mathbf{x}) - y\right)^2, \tag{8.19}$$

однако использование такой формулы на практике невозможно, так как конкретный вид $p(\mathbf{x})$ и $p(y|\mathbf{x})$ не известен .[22]

Наивная замена интегрирования по распределению на суммирование по учебной выборке не корректна из-за эффекта *переобучения* при котором модель тем или иным способом просто запоминает учебную выборку (т.е. преобразует её в свои коэффициенты, а в качестве предсказания фактически выполняет поиск), но совершенно не обобщает закономерности, присутствующие в данных, и поэтому, как говорят, обладает слабой предсказательной силой. Поэтому для оценки метрик качества модели должна использоваться отдельная выборка, называемая *тестовой выборкой*, но полученная из того же распределения, что и учебная. На практике, все доступные данные случайным образом разделяются на две подвыборки: бóльшая часть используется для обучения, а меньшая откладывается до момента оценки качества модели.

Кроме того, осторожность следует соблюдать в процессе подбора гиперпараметров модели. Процедура подбора гиперпараметров по сути состоит

[21] *training set*

[22] Кроме того, не всегда даже понятно, что такое «настоящее распределение». Рост популярности применения методов машинного обучения для решение повседневных бытовых задач позволяет, например, задать следующий немаловажный вопрос. Можно ли найти вектор признаков $\mathbf{x}^\dagger$, или даже искусственно сконструировать объект, обладающий таковыми признаками, такой, что в ответ на этот вектор признаков готовая модель будет предсказывать требуемый для злоумышленника результат $y^\dagger$, т.е. $f(\mathbf{x}^\dagger) = y^\dagger$? Интуитивно понятно, что искать такой $\mathbf{x}^\dagger$ следует в области $p(\mathbf{x}) \approx 0$ или вне области покрытой тестовой выборкой.

в переборе значений гиперпараметров и оценке метрики качества модели с использованием так называемой *валидационной выборки*, которая имеет такие же статистические свойства как учебная и тестовая выборки. Валидационная выборка чаще всего получается разделением учебной выборки на непосредственно учебную и валидационную, подобно тому как это происходит с тестовой выборкой. Необходимость в отдельной выборке обусловлена тем, что процесс перебора значений гиперпараметров, независимо от того выполняется ли он вручную или с помощью специализированного алгоритма, представляет собой вариант алгоритма оптимизации, следовательно, потенциально может сложиться такая ситуация, в которой гиперпараметры подобраны таким образом, что модель *случайно* имеет высокую эффективность для валидационной выборки. Поэтому необходимо использовать тестовую выборку только для окончательной характеризации качества полученной модели с подобранными значениями гиперпараметров. Таки образом, на практике, все доступные данные случайным образом разделяются на три подвыборки.

В качестве простой альтернативы разбиению на учебную и валидационную выборки можно упомянуть *перекрёстную проверку k-ой кратности*[23]. Идея состоит в том, что доступная учебная выборка разбивается на k примерно одинаковых по размеру подмножеств данных. Важно, чтобы статистические свойства каждой подвыборки были одинаковыми, если необходимо, то следует выполнить случайное перемешивание данных перед разбиением. Полученные подмножества можно k способами скомбинировать назад в два новых набора данных: набор единичного размера, который будет выполнять роль валидационного, и набор, состоящий из остальных $k-1$ подмножеств, который будет выполнять роль учебной выборки. Для каждого из k способов комбинирования подмножеств следует обучить модель и вычислить требуемые метрики качества. Сразу видно, что недостаток метода перекрёстной проверки в том, что требуется в k раз больше вычислений — мы строим k моделей с одинаковым набором гиперпараметров, но используя немного разные наборы данных. Однако, вместо одного значения метрики качества мы получаем выборку из k реализаций метрики качества, поэтому мы можем вычислить не только выборочное среднее, но, например, и выборочную дисперсию. Иными словами, мы можем оценить сколько знаков в конкретном значении метрики качества статистически значимы. Если требуется сравнить между собой качество двух моделей и проверить, что какая-то одна лучше, то можно использовать тот или иной статистический критерий, например, для проверки равенства средних двух выборок (см. главу 7).

8.3 Методы на основе ансамблей деревьев

Даже не применяя формальные количественные меры эффективности, очевидно следующее неудовлетворительное поведение решающих деревьев. Неогра-

[23] *k-fold cross-validation*

ниченное в глубину решающее дерево разбивает пространство признаков на маленькие области, каждая из которых содержит одну точку данных, игнорируя присутствие погрешности во входных данных, и, таким образом, предсказательная мощь модели снижается за счёт того, что шум не усредняется по соседним областям пространства. Это пример переобучения применительно к методу решающего дерева.

При ограничении глубины дерева, напротив, пространство разбивается на несколько больших по размеру областей, таким образом аппроксимация получается слишком грубой и не учитывающей отдельные особенности.

Одним из элегантных способов преодоления этой проблемы являются методы, основанные на ансамблях деревьев. Ансамбль (т.е. набор) деревьев логично называют *лесом*. До настоящего момента алгоритм построения решающего дерева был полностью детерминирован и зависел лишь от входных данных. Однако, мы понимаем, что вектора признаков $\mathbf{x}_i$ сами по себе могут быть подвержены, например, погрешностям измерений, т.е. при повторном проведении идентичного эксперимента значения векторов будут отличаться в рамках точности измерений. А это повлечёт за собой изменение структуры дерева, и, как следствие, мы получим немного другую аппроксимирующую функцию. Понятно, что нас интересует именно усреднение таких индивидуальных аппроксимирующих функций для всех гипотетически-возможных погрешностей данных, так как такая модель хорошо различала бы случайный разброс данных из-за ошибки (и усредняла его) и систематическое поведение аппроксимируемой зависимости.

Однако, другой реализации набора данных у нас нет, поэтому попробуем сымитировать выборочный шум внося элемент случайности в построение дерева. При каждом новом выполнении процедуры будет получаться новая реализация дерева, т.е. выборка реализаций деревьев некоторого размера, или ансамбль. Очевидно, что для одного и того же $\mathbf{x}$ каждое дерево может теперь возвращать различный результат.

Сразу можно условиться, что для задачи регрессии финальным ответом нашей модели будет являться усреднённый ответ всех деревьев, а для задачи классификации мы будем подсчитывать долю деревьев, которые высказались за принадлежность точки к положительному классу, и зададим порог в соответствии с метриками, описанными в разделе 8.2. Так как от перестановки слагаемых сумма не меняется, видно, что каждое дерево в таком ансамбле независимо и равноправно. Существует и альтернативный подход, который называется усилением и рассматривается в разделе 8.3.5.

8.3.1 Bootstrap aggregation

Один из способов внести случайность — так называемый *bootstrap aggregation*[24], или сокращённо *bagging*. Этот метод впервые был предложен в работе американского исследователя Лео Бреймана (см. Leo Breiman 1996). Для по-

[24] К сожалению, общеупотребимый перевод термина на русский язык пока не успел сформироваться.

строения ансамбля деревьев используется *бутстрап*[25], общий подход, который имитирует выборочный шум с помощью построения подвыборок с возвратом. Для выборки из N пар $\mathbf{x}_i, y_i$ необходимо задать размер подвыборки M, а затем случайно выбрать M элементов из исходной выборки. Вероятность выбора каждого элемента составляет $1/N$ и не зависит от соответствующих значений $\mathbf{x}_i, y_i$. Поэтому можно говорить, что исходная выборка и все реализации случайной подвыборки имеют одинаковые статистические свойства, однако различную реализацию выборочного шума.

Для построения каждого дерева ансамбля берётся уникальное случайное подмножество исходных данных. С одной стороны, поскольку мы считаем, что случайные подвыборки меньшего размера статистически похожи на исходный набор данных, мы ожидаем, что все деревья смогут решить исходную задачу. С другой стороны, теперь каждое из решающих деревьев получится немного отличным, так как каждое из деревьев строится на своём случайном подмножестве данных. Вспоминая рисунок 8.2, можно сказать, что на плоскости это будет графически соответствовать тому, что прямоугольники разбиения будут расположены немного в разных местах и имеют различную форму, т.е. сегменты кусочно-постоянной функции будут флуктуировать случайным образом. За счёт последующего усреднения по ансамблю деревьев, результирующая модельная функция будет иметь уже гораздо больше отдельных участков постоянного значения, что позволяет более плавно описывать изменения аппроксимируемой зависимости, чем это удалось бы сделать с помощью отдельного дерева.

8.3.2 Случайный лес

Метод *случайного леса*[26] развивает подход bootstrap aggregation. Деревья, построенные по методу bootstrap aggregation оказываются скореллированными друг с другом. Так, например, разбиения на верхних уровнях почти всегда используют одни и те же признаки, хотя и выбирают немного разные пороги. Это связано с тем, что хоть выборки данных для каждого дерева и разные, объем выборок при начальных разбиениях настолько велик, что выборочный шум не может повлиять на результат решения задач типа (8.7) или (8.10). Действительно, достаточно очевидно, что дисперсия значений целевых функции (8.7) или (8.10) падает с ростом объёма выборки N, как это происходит, например, с ошибкой выборочного среднего.

Оказывается, что можно улучшить эффективность метода, внеся дополнительное разнообразие в получаемые модели (см. Leo Breiman 2001). Так, предлагается на каждом шаге выбирать подмножество признаков по которым разбиение доступно, а остальные признаки на данном шаге игнорируются. Формально говоря, на каждом шаге вводится ограничение для задач оптимизации типа (8.7) или (8.10) — $k_0 \in K$, где K некоторое случайное подмножество множества всех индексов $1, 2, ..., n$, n обозначает размерность пространства признаков.

[25] *bootstrap*
[26] *random forest*

8.3.3 Ансамбль случайных деревьев

Известны и другие способы ещё большей рандомизации деревьев. Например, *чрезвычайно случайные деревья*[27] строятся по следующему принципу (см. Geurts, Ernst и Wehenkel 2006). Как и в методе случайного леса для каждого разбиения выбирается некоторое случайное подмножество рассматриваемых признаков, но затем, в отличии от метода случайного леса, порог разбиения для каждого признака назначается случайным образом, необходимо лишь гарантировать, чтобы в каждое из потенциальных поддеревьев попала хотя бы одна точка данных. Понятно, что количество таких кандидатов является некоторым настроечным параметром. Затем, среди этих разбиений-кандидатов выбирается оптимальный в смысле оптимизации (8.7) или (8.10).

Итак, сравнивая методы bootstrap aggregation (см. раздел 8.3.1), случайного леса (см. раздел 8.3.2) и ансамбля случайных деревьев, можно заметить, что их основное отличие состоит в том, как сформулирован критерий разбиения, т.е. задачи оптимизации типа (8.7) или (8.10). Если в методе bootstrap aggregation на каждом шаге допускается поиск любого возможного разбиения, в методе случайного леса возмножные разбиения ограничиваются условием $k_0 \in K$, где K некоторое случайное подмножество всех индексов признаков, а в методе ансамбля случайных деревьев возможные разбиения ограничены несколькими случайными вариантами.

На первый взгляд удивительно, но оказывается, что даже такой ансамбль полностью случайных деревьев (т.е. когда для каждого разделения выбирается один случайный признак и один случайный порог) способен эффективно решать задачи классификации и регрессии. Очевидным преимуществом такого подхода перед методом случайного леса является скорость построения деревьев.

8.3.4 Изолирующий лес

Чрезвычайно случайные деревья, в которых на каждом этапе разбиение производится полностью случайным образом, практически независимо от значений входных данных[28], имеют ещё одного интересное применение — поиск вылетающих значений в выборке. Вылетающими значениями обычно называют такие элементов выборки, которые в пространстве параметров отстоят от соседей дальше, чем это принято в среднем, т.е. населяют области с низкой плотностью вероятности. В отличии от предыдущих задач, такую задачу называют задачей машинного обучения *без учителя*[29]. Отсутствие учителя означает отсутствие меток или набора y_i.

[27] *extremely randomized tree* или *extra-tree*

[28] Получаемые разбиения, тем не менее, не должны быть вырожденными, т.е. в левом и правом поддеревьях должен находиться хотя бы один элемент. Это означает, что порог разбиения $\hat{x}_0$ должен быть выбран между минимальным и максимальным значением признака. В этом смысле, зависимость разбиений от входных данных существует.

[29] *unsupervised*

Действительно, в случае, когда несколько раз измеряется одна и та же скалярная физическая величина (например, для последующего повышения точности, путём усреднения), известно так называемое «правило трёх сигм», позволяющее отбросить измерения, существенно отличные от среднего значения без рефлексии о происхождении таких измерений. Когда измерения представляют собой вектор, составленный из нескольких величин, то гипотеза о нормальности многомерного распределения часто оказывается ошибочной, и тогда наивный аналог правила трёх сигм не действует.

Можно было бы попытаться оценить плотность числа точек в различных областях многомерного пространства признаков, а затем исключить точки из областей с плотностью меньше пороговой. В одномерном случае, пожалуй самый простой способ получить оценку плотности вероятности состоит в построении гистограммы. Однако, в случае пространств с большой размерностью число требуемых бинов многомерной гистограммы растёт степенным образом в зависимости от размерности пространства, что делает процедуру строительства многомерной гистограммы неэффективной сразу с нескольких точек зрения. С вычислительной точки зрения нам потребуется очень большое количество памяти и вычислений, степенным образом зависящие от размерности пространства признаков. С точки зрения статистики в большинстве бинов либо не будет данных вообще, либо будет лишь несколько записей, так как бинов будет неизбежно много больше, чем данных.

Метод изолирующего леса (см. Liu, Ting и Zhou 2008) представляет собой эффективный способ разбиения многомерного пространства с последующей оценкой относительной плотности числа точек данных в каждой точке пространства $\mathbf{x}$. Изолирующий лес строится из чрезвычайно случайных деревьев, где каждое разбиение выбирается случайным образом. Понятно, что раз метки y_i больше не участвуют в постановке задачи, то оценить оптимальность разбиения по формулам (8.7) или (8.10) больше нельзя. Следовательно, имеет смысл генерировать только один случайный кандидат в разбиения, необходимо лишь гарантировать, что разбиение окажется невырожденным, т.е. в левом и правом поддереве окажется хотя бы одна точка. Для этого обычно вначале случайным образом с равной вероятностью выбирается индекс признака разбиения k_0, находятся минимальное и максимальное значения данного признака x_{k_0} из доступной выборки, которые определяют границы равномерного распределения из которого разыгрывается порог разбиения $\hat{x}_0$. Разбиение выборки продолжается до тех пор, пока каждая точка не окажется *изолирована* в отдельном листе.

Построенные таким образом деревья обладают двумя полезными свойствами для решения исходной задачи. Во-первых, средняя глубина любого поддерева пропорциональна логарифму числу листьев (т.е. точек исходной выборки) содержащихся в этом поддереве. Действительно, рассмотрим

среднюю глубину d одномерного случайного поддерева с N листьями:

$$d(N) = \frac{1}{N-1}\sum_{k=1}^{N-1}\left(\frac{k}{N}(d(k)+1) + \frac{N-k}{N}(d(N-k)+1)\right) = \\ = 1 + \frac{1}{N-1}\sum_{k=1}^{N-1}\left(\frac{k}{N}d(k) + \frac{N-k}{N}d(N-k)\right) = \\ = 1 + \frac{2}{N(N-1)}\sum_{k=1}^{N-1} kd(k). \quad (8.20)$$

Если принять $d(1) = 0$ и $d(2) = 1$, тогда решением этого разностного уравнения будет

$$d(N) = \sum_{k=2}^{N}\frac{2}{N} = 2\left(H(N) - 1\right), \quad (8.21)$$

где гармоническое число $H(N) \approx \gamma + \ln N$, где $\gamma \approx 0.5772$ называется постоянной Эйлера.

Во-вторых, средний объём области пространства признаков, приписанный каждому узлу, оказывается пропорционален $2^{-d(N)}$. В отличии от сбалансированных бинарных деревьев, где все листья имеют почти одинаковую глубину, в случае со случайными деревьями листья будут иметь существенно разную глубину. Поэтому глубокие листья соответствуют области пространства признаков с высокой плотностью точек, а листья с малой глубиной — области пространства с низкой плотностью точек.

Объяснить это можно и альтернативным способом. Допустим, что все значения x_i некоторого признака сосредоточены в одной области, и есть одно значение $\widetilde{x}$, отстоящее от этой группы на расстоянии много большим размера этой области. Мы хотим построить разбиение этого набора на две ветви поддерева, выбрав случайным образом из равномерного распределения порог $\hat{x}_0$. Так как расстояние между вылетающим значением и кластером значений много больше размера кластера, то пропорциональная этому пустому расстоянию вероятность выбрать порог между ними 1. Иными словами, вылетающая точка будет изолирована в своём персональном поддереве (левом или правом) с большой вероятностью, независимо от конкретного значения $\hat{x}_0$. Понятно, что поддерево содержащее одну точку невозможно разбить дальше и оно образует лист.

Когда мы строим ансамбль случайных деревьев, то для каждой точки пространства $\mathbf{x}$ лист, накрывающий эту точку в каждом дереве, будет очевидно иметь немного разную глубину. Усредним эту глубину по ансамблю:

$$\bar{d}(\mathbf{x}) \equiv \frac{1}{M}\sum_{m=0}^{M} d_m(\mathbf{x}). \quad (8.22)$$

На практике удобно ввести нормированную величину в следующем виде:

$$s(\mathbf{x}) = -2^{-\frac{\bar{d}(\mathbf{x})}{d(N)}}, \quad (8.23)$$

где $d(N)$ вычисляется по формуле (8.21). Следовательно, величина $s(\mathbf{x}) \to -1$ для $\bar{d}(\mathbf{x}) \to 0$, и наоборот $s(\mathbf{x}) \to +1$ для $\bar{d}(\mathbf{x}) \to +\infty$.

Таким образом для каждой точки $\mathbf{x}$ в пространстве признаков мы можем вычислить величину пропорциональную плотности числа точек применяя выражение (8.23). Значит можно подсчитать $s(\mathbf{x}_i)$ для каждой точки выборки с последующей сортировкой. Если заранее задать долю количества точек, которую допустимо убрать из исходной выборки ради её очищения от выбросов, то в отсортированной последовательности $s(\mathbf{x}_i)$ легко найти порог s_0, такой, что данные с $s(\mathbf{x}_i) \leq s_0$ признаются выбросами и удаляются из выборки.

8.3.5 Градиентное усиление над решающими деревьями

В предыдущих случаях все деревья строились независимо друг от друга, позволяя, кстати, почти неограниченно распараллеливать процесс построения леса между несколькими потоками исполнения. Такой подход часто называют *методом комитетов*. Существует альтернативный способ составления ансамблей деревьев, известный как *усиление*[30] (см. Friedman 2001). Усиление предполагает, что искомая функция представляется рядом

$$f(\mathbf{x}) = \sum_{m=0}^{M} f_m(\mathbf{x}), \tag{8.24}$$

где слагаемые $f_m(\mathbf{x})$ строятся последовательно на основе $f_0(\mathbf{x}), ..., f_{m-1}(\mathbf{x})$.

Градиентное усиление использует следующий подход для решения задачи регрессии. В качестве $f_0(\mathbf{x})$ строится ограниченное по глубине решающее дерево, затем подсчитывается величина

$$\mathbf{r}^{(0)} \equiv \mathbf{f}^{(0)} - \mathbf{y} = \frac{1}{2}\nabla_{\mathbf{f}^{(0)}} \|\mathbf{f}^{(0)} - \mathbf{y}\|^2, \tag{8.25}$$

где вектор $\mathbf{y}$ обозначает все метки, а вектор $\mathbf{f}^{(0)}$ обозначает предсказание модели для всех точек учебного набора данных:

$$\mathbf{f}^{(0)} \equiv \begin{bmatrix} f_0(\mathbf{x}_1) \\ f_0(\mathbf{x}_2) \\ \vdots \\ f_0(\mathbf{x}_N) \end{bmatrix}. \tag{8.26}$$

Таким образом, вектор $\mathbf{r}^{(0)}$ по сути является вектором ошибок нашей модели на учебном наборе данных. Второе равенство в уравнении (8.25) объясняет название метода.

В качестве $f_1(\mathbf{x})$ строится следующее решающее дерево, но не для задачи регрессии y_i по признакам $\mathbf{x}_i$, а для регрессии $r_i^{(0)}$ по признакам $\mathbf{x}_i$.

[30] *boosting*

Иначе говоря, каждое следующее дерево учится исправлять ошибки предыдущего. А итоговое предсказание модели вычисляется по формуле (8.24), которой можно придать смысл последовательного уточнения предсказания с помощью коррекций всё более и более высокого порядка.

8.4 Логистическая регрессия

Помимо моделей основанных на деревьях, понятие машинное обучение включает в себя множество разнообразных методов, наибольшую известность среди которых в последнее время, пожалуй, имеют методы нейронных сетей (см. раздел 8.5). Перед тем как переходить к их рассмотрению имеет смысл изучить другой метод, называемый методом *логистической регрессии*. Как будет видно далее, логистическая регрессия представляет собой один из простейших вариантов нейронной сети, который, тем не менее, имеет очевидный способ обобщения.

Метод логистической регрессии используется для решения задачи классификации. Далее сначала рассмотрим случай бинарной классификации, а затем легко обобщим его на случай нескольких классов.

Вспомним, что в разделе 5.1 рассматривались модели со скрытыми переменными и смеси распределений. Перепишем выражение (5.27) для условной вероятности принадлежности измерений тому или иному классу при условии известного вектора признаков $\mathbf{x}$:

$$p(\{z=j\}\,|\mathbf{x},\theta') = \frac{p(\mathbf{x}|\,\{z=j\}\,,\theta')p(\{z=j\}\,|\theta')}{\sum_{k\in\{0,1\}} p(\mathbf{x}|\,\{z=k\}\,,\theta')p(\{z=k\}\,|\theta')}. \tag{8.27}$$

В отличии от раздела 5.1, теперь мы считаем, что в рамках учебной выборки переменная y, обозначающая принадлежность измерений тому или иному классу, известна, т.е. больше не является скрытой. Параметры θ' и аналитический вид $p(\mathbf{x}|z,\theta')$ нам неизвестны и не интересны — в рамках данного подхода нас интересует только возможность вычислять (8.27) для любого вектора $\mathbf{x}$, получая вероятность принадлежности события определённому классу.

Для этого преобразуем (8.27) для $j=1$ в следующий вид, разделив обе части дроби на числитель:

$$p(\{z=1\}\,|\mathbf{x},\theta') = \frac{1}{1+\exp\left(-\ln\left(\frac{p(\mathbf{x}|\{z=1\},\theta')p(\{z=1\}|\theta')}{p(\mathbf{x}|\{z=0\},\theta')p(\{z=0\}|\theta')}\right)\right)}, \tag{8.28}$$

для удобства, обозначим логарифм в этом выражении новой переменной $v(\mathbf{x})$, и аппроксимируем эту неизвестную функцию линейным разложением:

$$p(\{z=1\}\,|\mathbf{x},\theta') = \frac{1}{1+\exp(-v(\mathbf{x}))}, \tag{8.29}$$

$$v(\mathbf{x}) = (\mathbf{x},\mathbf{w}) + b, \tag{8.30}$$

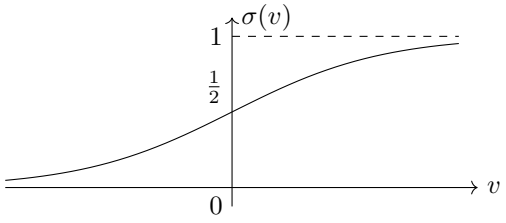

Рис. 8.5: График функции $\sigma(v) \equiv \frac{1}{1+\exp(-v)}$, называемой логистическим сигмоидом.

а из условия нормировки вероятности следует, что

$$p(\{z=0\}\,|\mathbf{x},\theta') = \frac{\exp(-v(\mathbf{x}))}{1+\exp(-v(\mathbf{x}))}, \tag{8.31}$$

где $\mathbf{w}$ и b — некоторые новые неизвестные параметры, которые уже не зависят от конкретного распределения. Функцию вида (8.29) называют *логистическим сигмоидом*, она монотонно возрастает от 0 до 1 и определена для всех значений v. График функции (8.29) представлен на рисунке 8.5. Корректность приближения (8.30) можно оправдать, рассматривая два нормальных распределения $p(\mathbf{x}|z,\theta')$ с одинаковыми матрицами ковариации, в таком случае линейность выражения может быть проверена строго, т.е. новые параметры $\mathbf{w}$ и b выразятся через традиционные параметры многомерного нормального распределения. В других случаях, (8.29) является эмпирической моделью.

Необходимо отыскать оценки новых параметров $\mathbf{w}$ и b, которые, в отличии от раздела 5.1, больше не представляют для нас самостоятельного интереса. Проще всего выполнить оценку с помощью метода максимального правдоподобия, так как для учебной выборки нам одновременно известны и метки y_i, определяющие принадлежность к классу, и вектора признаков x_i. Запишем отрицательный логарифм правдоподобия, который в этом случае называют *перекрёстной энтропией*:

$$\begin{aligned} &-\ln p(\{z_1=y_1, z_2=y_2, ..., z_N=y_N\}\,|\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_N,\theta) = \\ &= -\sum_{i=1}^{N}\left(y_i \ln p(\{z=1\}\,|\mathbf{x}_i,\theta) + (1-y_i)\ln\left(1-p(\{z=1\}\,|\mathbf{x}_i,\theta)\right)\right), \end{aligned} \tag{8.32}$$

где θ обозначает параметры $\mathbf{w}$ и b, а $y_i \in \{0,1\}$. Оценка параметров находится с помощью численных методов оптимизации (8.32) в соответствии с методом максимального правдоподобия (см. главу 5).

Подставляя выражения (8.29) и (8.31) в (8.32) и вычисляя производные по неизвестным параметрам $\mathbf{w}$ и b, можно получить следующую систему нелинейных алгебраических уравнений для вычисления оценки неизвест-

ных параметров $\hat{\mathbf{w}}$ и $\hat{b}$:

$$\sum_{i=1}^{N} \left(y_i - p(\{z=1\} \,|\mathbf{x}_i, \hat{\mathbf{w}}, \hat{b}) \right) = 0, \tag{8.33}$$

$$\sum_{i=1}^{N} \left(y_i - p(\{z=1\} \,|\mathbf{x}_i, \hat{\mathbf{w}}, \hat{b}) \right) \mathbf{x}_i = 0. \tag{8.34}$$

К сожалению, аналитического решения этой системы уравнений не существует, на практике решение находится численными методами.

В случае многоклассовой классификации вместо выражения (8.27) получается следующее:

$$p(\{z=j\} \,|\mathbf{x}, \theta') = \frac{\exp(v_j(\mathbf{x}))}{\sum_{k=1}^{m} \exp(v_k(\mathbf{x}))}, \tag{8.35}$$

$$v_j(\mathbf{x}) \equiv \ln p(\mathbf{x}|\, \{z=j\}, \theta) p(\{z=j\} \,|\theta), \tag{8.36}$$

где $j \in \{1, ..., m\}$ обозначает один из m целевых классов. Эта функция называется *нормированной экспонентой* или *softmax*. Затем, каждая функция $v_j(\mathbf{x})$ приближается линейной зависимостью с неизвестными параметрами $\mathbf{w}_j$ и b_j, которые находятся исходя из метода максимального правдоподобия, минимизацией перекрёстной энтропии:

$$-\ln p(\{z_1 = y_1, z_2 = y_2, ..., z_N = y_N\} \,|\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_N, \theta) = \\ = -\sum_{i=1}^{N} \sum_{j=1}^{m} h_{ij} \ln p(\{z=y_i\} \,|\mathbf{x}_i, \theta), \tag{8.37}$$

где θ обозначает все искомые параметры $\mathbf{w}_1, ..., \mathbf{w}_m$, $b_1, ..., b_m$, а h_{ij} обозначает представление метки класса в виде *унитарного кода*[31]:

$$h_{ij} \equiv \begin{cases} 1 & \text{если } y_i = j, \\ 0 & \text{иначе.} \end{cases} \tag{8.38}$$

8.5 Нейронные сети

8.5.1 Нейронные сети прямого распространения

Очевидный недостаток метода логистической регрессии (см. раздел 8.4) состоит в том, что линейное приближение не всегда хорошо работает для реальных распределений признаков. Для улучшения модели требуется предусмотреть возможность нелинейной аппроксимации $v(\mathbf{x})$. Метод нейронный сетей строит такую нелинейную аппроксимацию методом композиции функций (т.е. построения сложных функций) следующим образом. Представим, что теперь $v(\mathbf{x})$ задаётся следующим образом:

$$v(\mathbf{x}) = (\mathbf{w}_2, \mathbf{z}(\mathbf{x})) + b_2, \tag{8.39}$$

[31] *one-hot encoding*

где $\mathbf{z}(\mathbf{x})$ в свою очередь определяется как

$$\mathbf{z} = \varphi\left(W_1\mathbf{x} + b_1\right), \tag{8.40}$$

где W_1 — некоторая матрица параметров, а φ обозначает поэлементное применение к аргументу-вектору нелинейного преобразования, называемого *функцией активации*. Для простоты можно считать, что φ обозначает известную нам логистическую сигмоиду:

$$\left(\varphi(\mathbf{v})\right)_i = \frac{1}{1+\exp(-v_i)}, \tag{8.41}$$

однако на практике используются и другие варианты, такие, например, как гиперболический тангенс:

$$\left(\varphi(\mathbf{v})\right)_i = \mathrm{th}(v_i), \tag{8.42}$$

или линейный выпрямитель[32]:

$$\left(\varphi(\mathbf{v})\right)_i = \max\left\{0, v_i\right\}. \tag{8.43}$$

Уравнения (8.29), (8.47) и (8.40) вместе определяют простую модель нейронной сети прямого распространения с одним скрытым слоем:

$$p(\{z=1\}\,|\mathbf{x}, \theta') = \varphi((\mathbf{w}_2, \varphi\left(W_1\mathbf{x} + b_1\right)) + b_2), \tag{8.44}$$

где θ' обозначает параметры W_1, $\mathbf{w}_2$, b_1 и b_2. В терминологии нейронных сетей каждый компонент вектора признаков $\mathbf{x}$, вектора $\mathbf{z}$ и вероятность $p(\{z=1\}\,|\mathbf{x}, \theta')$ принято называть *нейроном*. Нейроны объединяются в *слои*, и тогда вектор $\mathbf{x}$ называют *входным слоем*, вектор $\mathbf{z}$ *скрытым слоем*, а предсказываемую вероятность — *выходным слоем*. В случае бинарной классификации выходной слой состоит из одного нейрона, в случае многоклассовой классификации выходной слой чаще всего представляется с помощью вектора вероятностей точно так же как в случае логистической регрессией. Понятно, что число скрытых слоёв и их размер являются гиперпараметрами. Кроме того, возможно несколько вариантов выбора функции активации, особенно для промежуточных слоев.

Так как запись вида (8.44) выглядит громоздко, а все расчёты выполняются численно, то на практике она применяется редко, а нейронные сети принято изображать графически. Пример графического изображения нейронной сети (8.44) приведён на рисунке 8.6. Нейронные сети, подобные изображённой на рисунке, называют сетями *прямого распространения*, так как все связи (стрелки) направлены только слева направо. Кроме того, нейронные сети, у которых задействованы все возможные связи между соседними слоями, называют *полносвязными сетями*. Полносвязные нейронные сети прямого распространения — популярная, но далеко не единственная известная архитектура (см., например, Хайкин 2018).

[32] ReLU (Rectified linear unit)

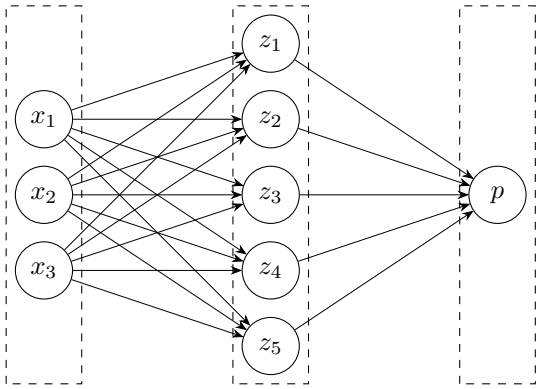

Рис. 8.6: Графическое изображение нейронной сети, состоящей из трёх слоёв: входного $\mathbf{x} \in \mathrm{R}^3$, скрытого (промежуточного) $\mathbf{z} \in \mathrm{R}^5$ и выходного $p = p(\{z = 1\} \,|\mathbf{x}, \theta')$. Стрелками обозначены связи между нейронами соседних слоёв. Штриховыми линиями обведены нейроны, принадлежащие к одному слою.

Параметры θ' называют *весами* и они всё ещё подлежат определению исходя из учебной выборки данных и минимизации выбранной *функции потерь*. Для задачи классификации функция потерь выбирается в виде (8.32) или (8.37) в зависимости от рассматриваемого числа классов. Процесс оценки неизвестных весов с помощью алгоритмов численной оптимизации называют *обучением нейронной сети*, важные практические детали численной оптимизации для нейронных сетей рассматриваются в разделе 8.5.2.

Задача регрессии

Несмотря на то, что мы мотивировали метод нейронных сетей недостатками в методе логистической регрессии, нейронные сети могут применяться не только для решения задач классификации, но и других задач, например, задачи регрессии. Более того, оказывается, что существуют строгие теоретические результаты (см. Колмогоров 1957 и Горбань 1998), формально показывающие, что с помощью подхода нейронных сетей можно построить точную аппроксимацию любой непрерывной функции многих переменных, при ключевом условии, что $\varphi(\mathbf{v})$ — нелинейная функция.

В случае задачи классификации, нейронная сеть задавала способ вычисления $p(\{z = 1\} \,|\mathbf{x})$. В случае задачи регрессии, нейронная сеть задаёт способ вычисления некоторой функции $f(\mathbf{x})$, которая, как мы считаем, может использоваться для предсказания значения меток y. Для этого, во-первых, требуется выбрать функцию активации для выходного слоя таким образом, чтобы она соответствовала ожидаемому диапазону меток y, либо отнормировать данные соответствующим образом. Во-вторых, нужно выбрать адекватную задачи функцию потерь, например среднеквадратичное

отклонение:

$$\text{MSE} \equiv \frac{1}{N}\sum_{i=1}^{N}\left(f(\mathbf{x}_i|\theta) - y_i\right)^2, \tag{8.45}$$

где y_i обозначают метки учебной выборки, соответствующие признакам $\mathbf{x}_i$, а $f(\mathbf{x}|\theta)$ обозначает нейронную сеть целиком, параметры θ определяются в процессе обучения. При выборе такой функции потерь, задача обучения нейронной сети соответствует задаче нелинейных наименьших квадратов (4.24).

Задача уменьшения размерности

Кроме того, нейронные сети применяются и для решения задач машинного обучения без учителя, например, для задачи уменьшения размерности. *Автокодировщик* или *самоассоциативная нейронная сеть* — предназначенная для этого популярная архитектура сети. Автокодировщик представляет собой сеть прямого распространения, в которой входной и выходной слои имеют одинаковый размер, а один из скрытых слоёв имеет меньший размер. Тогда первая половина сети называется кодировщиком, а вторая декодировщиком. Функция потерь выбирается в следующем виде (сравните с выражением (3.18)):

$$R(\theta) = \frac{1}{N}\sum_{i=1}^{N}\|\mathbf{f}(\mathbf{x}_i|\theta) - \mathbf{x}\|^2, \tag{8.46}$$

где метки y_i в рассмотрении не участвуют, так как решается задача обучения без учителя, т.е. без меток. Фактически, мы пытаемся аппроксимировать тождественное преобразование, однако, малый размер скрытого слоя в середине сети мешает сделать это тривиальным способом. По принципу построения нейронных сетей выходной слой $\mathbf{f}(\mathbf{x}_i|\theta)$ неявно зависит от скрытого слоя меньшей размерности $\mathbf{z}$, который в свою очередь зависит от входных данных. Таким образом кодировщик должен научиться упаковывать представленный на вход вектор $\mathbf{x}$ в пространство меньшей размерности с потерей минимального количества информации, а декодировщик нужен чтобы кодировщик мог обучаться. В каком-то смысле декодировщик проверяет, что информация может быть преобразована в исходную форму.

После обучения первую часть нейронной сети, кодировщик, используют отдельно для решения задачи понижения размерности. Такая модель может использоваться как готовый блок для решения других задач, например классификации или регрессии. Вместо построения нейронной сети, которая принимает на вход исходный вектор признаков, строится нейронная сеть, которая принимает на вход сжатое представление. Такая сеть будет иметь меньше параметров, следовательно требовать меньших вычислительных ресурсов и данных для своего обучения. Автокодировщик способен решать задачу фильтрации случайных шумов, подобно анализу главных компонент (см. главу 3).

8.5.2 Обучение на основе коррекции ошибок. Обратное распространение ошибки

Рассмотрим важный вопрос о способе выполнения численной оптимизации над функциями, заданными с помощью нейронных сетей. Любой метод, кроме всего прочего, должен быть практически реализуем, т.е. все величины должны быть рассчитаны за разумное время, иначе от него будет не слишком много практической пользы. Понятно, что расчёт предсказания нейронной сети требует примерно одинаковое количество умножений и сложений, пропорциональное числу параметров модели, что кажется достаточно разумным. В таком случае расчёт предсказания выполняется послойно — от входного слоя к выходному.

Для эффективного выполнения численной оптимизации требуется рассчитать градиент функции потерь по искомым параметрам, такой вектор градиента сам будет иметь размер равный числу искомых параметров сети. К счастью, оказывается, что есть эффективный способ вычисления градиента, называемый *методом обратного распространения ошибки*, позволяющий вычислять градиент, используя для этого количество операций, линейно пропорциональное количеству параметров.

Итак, пусть каждый слой задаётся следующими уравнениями:

$$\mathbf{v}^{(l)} = W_l \mathbf{z}^{(l)} + \mathbf{b}_l, \tag{8.47}$$

$$\mathbf{z}^{(l+1)} = \varphi\left(\mathbf{v}^{(l)}\right). \tag{8.48}$$

Для однородности изложения $\mathbf{z}^{(0)}$ обозначает входной слой $\mathbf{x}$, а соответствующий $\mathbf{z}^{(n+1)}$ обозначает выходной слой нейронной сети. Пусть задана некоторая функция потерь $E(\mathbf{z}^{(n+1)})$, например в виде (8.32), (8.37), (8.45) или (8.46), тогда требуется вычислить частные производные этой функции по искомым параметрам: элементам матриц W_1, W_2, ..., W_n, и компонентам векторов $\mathbf{b}_1$, $\mathbf{b}_2$, ..., $\mathbf{b}_n$.

Прежде всего заметим, что функция потерь зависит от некоторого параметра только через значение соответствующего слоя, т.е., например, справедливо:

$$\frac{\partial E}{\partial W_{ij}^{(l)}} = \frac{\partial E}{\partial v_i^{(l)}} \frac{\partial v_i^{(l)}}{\partial W_{ij}^{(l)}} = \frac{\partial E}{\partial v_i^{(l)}} z_j^{(l)}, \tag{8.49}$$

$$\frac{\partial E}{\partial b_i^{(l)}} = \frac{\partial E}{\partial v_i^{(l)}} \frac{\partial v_i^{(l)}}{\partial b_i^{(l)}} = \frac{\partial E}{\partial v_i^{(l)}}. \tag{8.50}$$

Иначе говоря, рассчитав частные производные функции потерь по значениям слоёв, мы могли бы сравнительно легко пересчитать такие производные в нужные производные по параметрам. Для этого достаточно будет временно запомнить значения $\mathbf{z}^{(l)}$ для каждого слоя, которые вычисляются во время вычисления значения функции потерь.

Чтобы вычислить частные производные функции потерь E по компонентам векторов $\mathbf{v}^{(l)}$ заметим, что функция потерь зависит от значений на слое l только косвенно через значение слоя $l+1$ (см. рисунок 8.6):

$$\frac{\partial E}{\partial v_i^{(l)}} = \sum_k \frac{\partial E}{\partial v_k^{(l+1)}} \frac{\partial v_k^{(l+1)}}{\partial v_i^{(l)}} = \sum_k \frac{\partial E}{\partial v_k^{(l+1)}} w_{ki}^{(l)} \phi' \left(v_i^{(l)} \right). \tag{8.51}$$

Если представить, что при подсчёте функции потерь соответствующие значения слоёв будут временно записаны в память, то полученные уравнения позволяют рассчитать значения всех производных рекуррентно от выходного слоя к входному. Этим и объясняется название метода обратного распространения ошибки: расчёт предсказания производится пошагово в прямом направлении, а производной — в обратном. Частные производные функции потерь E по компонентам векторов $\mathbf{v}^{(n)}$ вычисляются непосредственным дифференцированием функции потерь, таким образом задавая начальные условия для рекуррентного расчёта. К счастью, популярные пакеты для работы нейросетевыми моделями выполняют автоматическое дифференцирование, и избавляют от необходимости программирования вычисления градиента, так как градиент вычисляется автоматически для любой заданной архитектуры сети.

При обучении нейронных сетей популярно применение алгоритмов типа стохастического градиентного спуска (см. раздел 4.4.1), при котором на каждом шаге используется лишь малая часть доступных данных. Такой способ обучения называется последовательным. Для каждой отдельной записи или небольшого набора, называемого *пакетом*[33], вычисляется оценка градиента, а затем веса подправляются на некоторую величину в направлении антиградиента:

$$\theta^{(k+1)} = \theta^{(k)} - \nu \nabla_\theta E, \tag{8.52}$$

где ν называется *скоростью обучения*. Скорость обучения зачастую подбирается вместе с остальными гиперпараметрами. Когда весь набор данных последовательно использован для оценки градиентов, говорят, что прошла одна эпоха обучения. Одна эпоха соответствовала бы одному шагу оптимизации при использовании пакетного обучения, т.е. когда градиент вычислялся бы на всех доступных данных сразу. Зависимость значения функции потерь от эпохи обучения называется *кривой обучения*.

Отдельно стоит упомянуть проблему выбора начальных значений весов. Пространство весов обладает большим числом симметрий, так например, все нейроны в рамках одного полносвязного слоя равноправны, поэтому задание весам одинаковых значений (например нулевых) приведёт к тому, что в процессе оптимизации эти значения будут обновляться на одинаковую величину. Понятно, что при этом нейронная сеть не сможет качественно обучиться, и чтобы избежать этого, начальные веса на практике выбираются случайным образом.

[33] *batch*

Список литературы

Armstrong, Joe (2003). “Making reliable distributed systems in the presence of software errors”. Дис. ... док. Royal Institute of Technology, Stockholm, Sweden. URL: https://erlang.org/download/armstrong_thesis_2003.pdf.

Bellm, Eric C. и др. (дек. 2018). “The Zwicky Transient Facility: System Overview, Performance, and First Results”. В: *Publications of the Astronomical Society of the Pacific* 131.995, с. 018002. DOI: 10.1088/1538-3873/aaecbe.

Bentley, Jon Louis (сент. 1975). “Multidimensional Binary Search Trees Used for Associative Searching”. В: *Commun. ACM* 18.9, с. 509—517. ISSN: 0001-0782. DOI: 10.1145/361002.361007.

Best, Michael J и Nilotpal Chakravarti (1990). “Active set algorithms for isotonic regression; a unifying framework”. В: *Mathematical Programming* 47.1-3, с. 425—439.

Breiman, L. и др. (1984). *Classification and Regression Trees*. Chapman и Hall/CRC. ISBN: 9780412048418.

Breiman, Leo (сент. 1996). “Bagging Predictors”. В: *Machine Learning* 24.2, с. 123—140. ISSN: 1573-0565. DOI: 10.1023/A:1018054314350.

— (окт. 2001). “Random Forests”. В: *Machine Learning* 45.1, с. 5—32. ISSN: 1573-0565. DOI: 10.1023/A:1010933404324.

Conn, A.R., N.I.M. Gould и P.L. Toint (2000). *Trust Region Methods*. MOS-SIAM Series on Optimization. Society for Industrial и Applied Mathematics. ISBN: 9780898714609.

Coppersmith, Don и Shmuel Winograd (1990). “Matrix multiplication via arithmetic progressions”. В: *Journal of Symbolic Computation* 9.3. Computational algebraic complexity editorial, с. 251—280. ISSN: 0747-7171. DOI: 10.1016/S0747-7171(08)80013-2.

Dempster, A. P., N. M. Laird и D. B. Rubin (1977). “Maximum Likelihood from Incomplete Data via the EM Algorithm”. В: *Journal of the Royal Statistical Society* 39 (1), с. 1—38.

Friedman, Jerome H. (2001). “Greedy function approximation: A gradient boosting machine.” В: *The Annals of Statistics* 29.5, с. 1189—1232. DOI: 10.1214/aos/1013203451.

Gaia Collaboration и др. (2023). “Gaia Data Release 3 - Summary of the content and survey properties”. В: *A&A* 674, A1. DOI: 10.1051/0004-6361/202243940.

Geurts, Pierre, Damien Ernst и Louis Wehenkel (апр. 2006). “Extremely randomized trees”. В: *Machine Learning* 63.1, с. 3—42. ISSN: 1573-0565. DOI: 10.1007/s10994-006-6226-1.

Guibas, Leo J. и Robert Sedgewick (1978). “A dichromatic framework for balanced trees”. В: *19th Annual Symposium on Foundations of Computer Science (sfcs 1978)*, с. 8—21. DOI: 10.1109/SFCS.1978.3.

Henze, Norbert (2002). “Invariant tests for multivariate normality: a critical review”. В: *Statistical Papers* 43, с. 467—506. DOI: 10.1007/s00362-002-0119-6.

Kingma, Diederik P. и Jimmy Ba (2017). *Adam: A Method for Stochastic Optimization*. DOI: 10.48550/arXiv.1412.6980. arXiv: 1412.6980 [cs.LG].

Lanczos, C. (1950). “An iteration method for the solution of the eigenvalue problem of linear differential and integral operators”. В: *Journal of Research of the National Bureau of Standards* 45 (4), с. 255. DOI: 10.6028/jres.045.026.

Levenberg, Kenneth (1944). “A Method for the solution of certain non – linear problems in least squares”. В: *Quarterly of Applied Mathematics* 2, с. 164—168. DOI: 10.1090/qam/10666.

Liu, Fei Tony, Kai Ming Ting и Zhi-Hua Zhou (2008). “Isolation Forest”. В: *2008 Eighth IEEE International Conference on Data Mining*, с. 413—422. DOI: 10.1109/ICDM.2008.17.

Marquardt, Donald W. (1963). “An Algorithm for Least-Squares Estimation of Nonlinear Parameters”. В: *Journal of the Society for Industrial and Applied Mathematics* 11.2, с. 431—441. DOI: 10.1137/0111030.

Metropolis, Nicholas и др. (июнь 1953). “Equation of State Calculations by Fast Computing Machines”. В: *The Journal of Chemical Physics* 21.6, с. 1087—1092. ISSN: 0021-9606. DOI: 10.1063/1.1699114.

Otsu, Nobuyuki (1979). “A Threshold Selection Method from Gray-Level Histograms”. В: *IEEE Transactions on Systems, Man, and Cybernetics* 9.1, с. 62—66. DOI: 10.1109/TSMC.1979.4310076.

Rao, C. Radhakrishna и др. (2007). *Linear Models and Generalizations: Least Squares and Alternatives*. Springer Series in Statistics. Springer Berlin. ISBN: 978-3-540-74226-5. DOI: 10.1007/978-3-540-74227-2.

Rasmussen, Carl Edward и Christopher K. I. Williams (2006). *Gaussian Processes for Machine Learning*. MIT Press. ISBN: 026218253X. URL: https://gaussianprocess.org/gpml/chapters/RW.pdf.

Robbins, Herbert и Sutton Monro (1951). “A Stochastic Approximation Method”. В: *The Annals of Mathematical Statistics* 22.3, с. 400—407. DOI: 10.1214/aoms/1177729586.

Schneider, J. и S. Kirkpatrick (2007). *Stochastic Optimization*. Scientific Computation. Springer Berlin. ISBN: 9783540345602.

Sedgewick, Robert и Philippe Flajolet (2014). *An Introduction to the Analysis of Algorithms*. 2nd Edition. CreateSpace Independent Publishing Platform. ISBN: 9781502575869.

Spall, J.C. (2005). *Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control*. Wiley Series in Discrete Mathematics and Optimization. Wiley. ISBN: 9780471441908.

Stigler, Stephen M. (1981). "Gauss and the invention of least squares". В: *The Annals of Statistics* 9 (3), с. 465—474.

Strassen, Volker (авг. 1969). "Gaussian elimination is not optimal". В: *Numerische Mathematik* 13.4, с. 354—356. DOI: 10.1007/bf02165411.

Turing, A. M. (1937). "On Computable Numbers, with an Application to the Entscheidungsproblem". В: *Proceedings of the London Mathematical Society* s2-42.1, с. 230—265. DOI: 10.1112/plms/s2-42.1.230.

Адельсон-Вельский, Г. М. и Е. М. Ландис (1962). "Один алгоритм организации информации". В: *Доклады АН СССР* 146 (2), с. 8—21.

Бишоп, Кристофер (2020). *Распознавание образов и машинное обучение*. Диалектика. ISBN: 978-5-907144-55-2.

Бокс, Д. и Г. Дженкинс (1974). *Анализ временных рядов: прогноз и управление*.

Вирт, Никлаус (2010). *Алгоритмы и структуры данных*. ДМК. ISBN: 978-5-94074-584-6.

Горбань, А.Н. (1998). "Обобщенная аппроксимационная теорема и точное представление многочленов от нескольких переменных суперпозициями многочленов от одного переменного". В: *Известия высших учебных заведений. Математика* 5, с. 6—9.

Кельберт, М.Я. и Ю.М. Сухов (2017). *Вероятность и статистика в примерах и задачах*. издание 3-е, дополненное. Т. 1. Основные понятия теории вероятностей и математической статистики. МЦНМО. ISBN: 978-5-4439-2326-0.

Клеппман, Мартин (2019). *Высоконагруженные приложения. Программирование, масштабирование, поддержка*. Питер. ISBN: 978-5-4461-0512-0.

Кнут, Дональд Эрвин (2019a). *Искусство программирования*. Т. 1. Основные алгоритмы. Вильямс. ISBN: 978-5-8459-1984-7.

— (2019b). *Искусство программирования*. Т. 2. Получисленные алгоритмы. Диалектика. ISBN: 978-5-8459-0081-4.

— (2019c). *Искусство программирования*. Т. 3. Сортировка и поиск. Диалектика. ISBN: 978-5-907144-41-5.

Колмогоров, А.Н. (1957). "О представлении непрерывных функций нескольких переменных в виде суперпозиций непрерывных функций одного переменного и сложения". В: *Доклады Академии наук СССР* 114 (5), с. 953—956.

Кормен, Томас и др. (2019). *Алгоритмы. Построение и анализ*. 3-е издание. Диалектика. ISBN: 978-5-907114-11-1.

Кун, Макс и Кьелл Джонсон (2019). *Предиктивное моделирование на практике*. Питер. ISBN: 978-5-4461-1039-1.

Лоусон, Ч. и Р. Хенсон (1986). *Численное решение задач метода наименьших квадратов*. Наука.

Мейер, Давид (1987). *Теория реляционных баз данных*. Мир.

Петров, Алекс (2021). *Распределенные данные. Алгоритмы работы современных систем хранения информации*. Питер. ISBN: 978-5-4461-1640-9.

Степанов, Сергей (2012). *Стохастический мир*. `https://synset.com/pdf/ito.pdf`.

Сухарев, А.Г., А.В. Тимохов и В.В. Федоров (2008). *Курс методов оптимизации*. издание 2-е. Физматлит. ISBN: 978-5-9221-0559-0.

Тихонов, А.Н. и В.Я. Арсенин (1986). *Методы решения некорректных задач*. 3-е издание, исправленное. Наука.

Хайкин, Саймон (2018). *Нейронные сети: полный курс*. 2-е, исправленное. Вильямс. ISBN: 978-5-8459-2069-0.

Оглавление