

Урок 1

«Знакомство с машинным обучением»

1.1. Знакомство с машинным обучением

Приветствуем Вас на курсе «обучение на размеченных данных». Это второй курс специализации «Машинное обучение и анализ данных», в котором начинается знакомство собственно с машинным обучением. Центральной темой этого курса является обучение с учителем. На самом деле эта тема была уже затронута в прошлом курсе, когда речь шла про интерполяцию. Интерполяция — задача восстановления функции по нескольким точкам, в которых известны ее значения.

Обучение с учителем — тоже восстановление общей закономерности по конечному числу примеров. Хотя постановки задач похожи, у них есть много отличий в том, как они решаются и какие требования выдвигаются к решению. Эти различия будут обсуждаться в данном курсе.

1.1.1. Пример: понравится ли фильм пользователю

Для начала будет рассмотрен не очень сложный пример, на котором можно понять, в чем заключается суть обучения с учителем и машинного обучения. Пусть есть некоторый сайт, посвященный кино, на который можно зайти, найти страницу нужного фильма, прочитать информацию про него: когда он снят, кто в нем играет и какой бюджет у этого фильма, а также, возможно, купить его и посмотреть. Пусть есть некоторые пользователи, которые находят страницу нужного фильма, читают и задаются вопросом «смотреть или нет?». Необходимо понять, понравится ли пользователю фильм, если выдать ему рекомендацию о фильме. Есть несколько подходов к решению:

- **Подход первый**, самый глупый — дать пользователю посмотреть этот фильм.
- **Второй подход** — дать случайный ответ и показать случайную рекомендацию. В обоих случаях пользователь может быть разочарован фильмом и он будет недоволен сайтом.
- **Третий подход** — пригласить психолога-киномана, чтобы разрешить ситуацию. Этот человек оценит пользователя, оценит фильм и поймет, понравится ли этот фильм этому пользователю, сопоставив информацию. Этот подход довольно сложный. Скорее всего таких специалистов не очень много, и будет сложно отмасштабировать это решение на миллионы пользователей сайта. Но на самом деле это не нужно.

Существует множество примеров — ситуаций, когда другие пользователи заходили на страницы фильмов, принимали решение посмотреть фильм и далее ставили оценку, по которой можно понять, понравился им фильм или нет. Задача машинного обучения состоит в восстановлении общей закономерности из информации в этих примерах.

1.1.2. Основные обозначения

В рамках данного курса будут использоваться следующие обозначения: x — объект, \mathbb{X} — пространство объектов, $y = y(x)$ — ответ на объекте x , \mathbb{Y} — пространство ответов.

Объектом называется то, для чего нужно сделать предсказание. В данном примере объектом является пара (пользователь, фильм). Пространство объектов — это множество всех возможных объектов, для ко-

торых может потребоваться делать предсказание. В данном примере это множество всех возможных пар (пользователь, фильм).

Ответом будет называться то, что нужно предсказать. В данном случае ответ — понравится пользователю фильм или нет. Пространство ответов, то есть множество всех возможных ответов, состоит из двух возможных элементов: -1 (пользователю фильм не понравился) и $+1$ (понравился).

Признаковым описанием объекта называется совокупность всех признаков:

$$x = (x^1, x^2, \dots, x^d).$$

Признак - это число, характеризующее объект. Признаковое описание является d -мерным вектором.

1.1.3. Выборка, алгоритм обучения

Центральным понятием машинного обучения является обучающая выборка $X = (x_i, y_i)_{i=1}^\ell$. Это те самые примеры, на основе которых будет строиться общая закономерность. Отдельная задача — получение обучающей выборки. В вышеупомянутом случае y_i — это оценка фильма пользователем.

Предсказание будет делаться на основе некоторой модели (алгоритма) $a(x)$, которая представляет из себя функцию из пространства \mathbb{X} в пространство \mathbb{Y} . Эта функция должна быть легко реализуема на компьютере, чтобы ее можно было использовать в системах машинного обучения. Примером такой модели является линейный алгоритм:

$$a(x) = \text{sign}(w_0 + w_1 x^1 + \dots + w_d x^d).$$

Операция взятия знака берется ввиду того, что пространство \mathbb{Y} состоит из двух элементов.

Не все алгоритмы подходят для решения задачи. Например константный алгоритм $a(x) = 1$ не подходит. Это довольно бесполезный алгоритм, который вряд ли принесет пользу сайту.

Поэтому вводится некоторая характеристика качества работы алгоритма — функционал ошибки. $Q(a, X)$ — ошибка алгоритма a на выборке X . Например, функционал ошибки может быть долей неправильных ответов. Следует особо отметить, что Q называется функционалом ошибки, а не функцией. Это связано с тем, что первым его аргументом является функция.

Задача обучения состоит в подборе такого алгоритма a , для которого достигается минимум функционала ошибки. Лучший в этом смысле алгоритм выбирается из некоторого семейства \mathbb{A} алгоритмов.

1.1.4. Решающие пни

Простейшим примером семейства алгоритмов являются решающие пни:

$$\mathbb{A} = \{ [x^j < t] \mid \forall j, t \}.$$

Здесь квадратные скобки соответствуют так называемой нотации Айверсона. Если логическое выражение внутри этих скобок — истина, то значение скобок равно 1, в ином случае — нулю.

Алгоритм работает следующим образом. Если значение определенного признака x^j меньше некоторого порогового значения t , то данный алгоритм возвращает ответ 0 (фильм не понравился), в ином случае — $+1$ (пользователю фильм понравился).

Решающие пни могут быть использованы для построения сложных композиций алгоритмов.

1.2. Обучение на размеченных данных

1.2.1. Постановка задачи

В этом разделе речь пойдет о том, какие бывают типы задач при обучении на размеченных данных, или обучении с учителем. Общая постановка задачи обучения с учителем следующая. Для обучающей выборки $X = (x_i, y_i)_{i=1}^\ell$ нужно найти такой алгоритм $a \in \mathbb{A}$, на котором будет достигаться минимум функционала ошибки:

$$Q(a, X) \rightarrow \min_{a \in \mathbb{A}}.$$

В зависимости от множества возможных ответов \mathbb{Y} , задачи делятся на несколько типов.

1.2.2. Задача бинарной классификации

В задаче бинарной классификации пространство ответов состоит из двух ответов $\mathbb{Y} = \{0, 1\}$. Множество объектов, которые имеют один ответ, называется классом. Говорят, что нужно относить объекты к одному из двух классов, другими словами, классифицировать эти объекты.

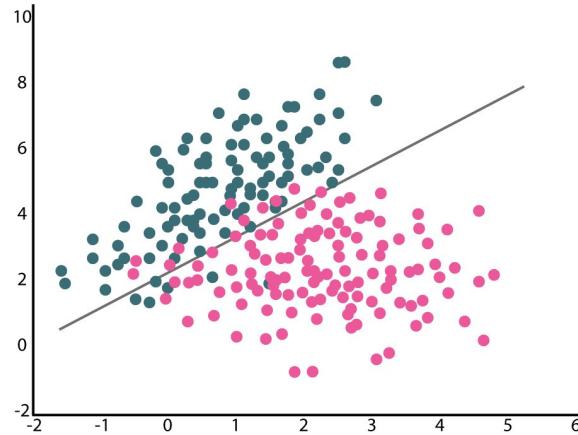


Рис. 1.1: Задача бинарной классификации

Примеры задач бинарной классификации:

- Понравится ли пользователю фильм?
- Вернет ли клиент кредит?

1.2.3. Задача многоклассовой классификации

Классов может быть больше, чем два. В таком случае имеет место задача многоклассовой классификации.

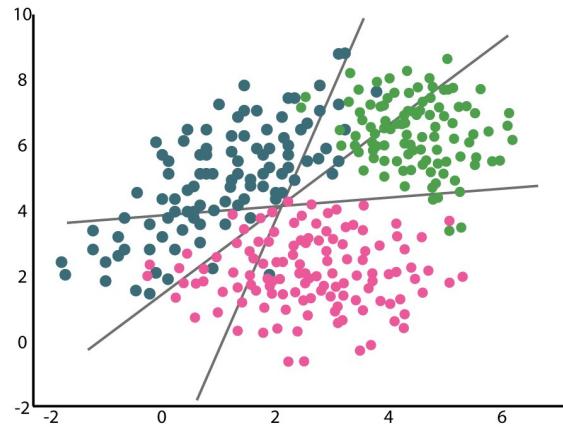


Рис. 1.2: Задача многоклассовой классификации

Примеры задач многоклассовой классификации:

- Из какого сорта винограда сделано вино?
- Какая тема статьи?
- Машина какого типа изображена на фотографии: мотоцикл, легковая или грузовая машина?

1.2.4. Задача регрессии

Когда y является вещественной переменной, говорят о задаче регрессии.

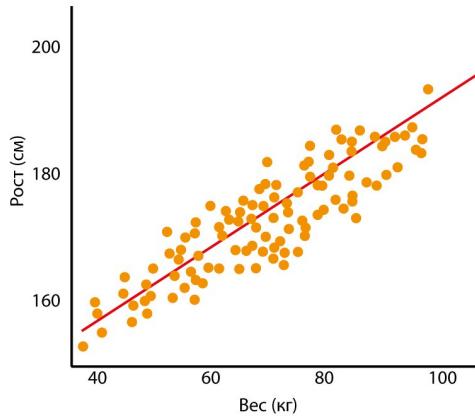


Рис. 1.3: Задача регрессии

Примеры задач регрессии:

- Предсказание температуры на завтра.
- Прогнозирование выручки магазина за год.
- Оценка возраста человека по его фото.

1.2.5. Задача ранжирования

Еще одним примером задачи обучения с учителем является задача ранжирования. Эта задача довольно тяжелая, и речь о ней в данном курсе не пойдет, но знать о ней полезно. Мы сталкиваемся с ней каждый день, когда ищем что-либо в интернете. После того, как мы ввели запрос, происходит ранжирование страниц по релевантности их запросу, то есть для каждой страницы оценивается ее релевантность в виде числа, а затем страницы сортируются по убыванию релевантности. Задача состоит в предсказании релевантности для пары (запрос, страница).

1.3. Обучение без учителя

В этом разделе мы обсудим, какие бывают постановки задач машинного обучения, кроме обучения с учителем.

Обучением с учителем называются такие задачи, в которых есть и объекты, и истинные ответы на них. И нужно по этим парам восстановить общую зависимость. Задача обучения без учителя — это такая задача, в которой есть только объекты, а ответов нет. Также бывают «промежуточные» постановки. В случае частичного обучения есть объекты, некоторые из которых с ответами. В случае активного обучения получение ответа обычно очень дорого, поэтому алгоритм должен сначала решить, для каких объектов нужно узнать ответ, чтобы лучше всего обучиться.

Рассмотрим несколько примеров постановки задач без учителя.

1.3.1. Задача кластеризации

Первый пример — задача кластеризации. Дано множество объектов. Необходимо найти группы похожих объектов. Есть две основные проблемы: не известно количество кластеров и не известны истинные кластеры, которые нужно выделять. Поэтому задача решается очень тяжело — здесь невозможно оценить качество решения. Этим и отличается задача классификации — там тоже нужно делить объекты на группы, но в классификации группы, а точнее классы, фиксированы, и известны примеры объектов из разных групп.

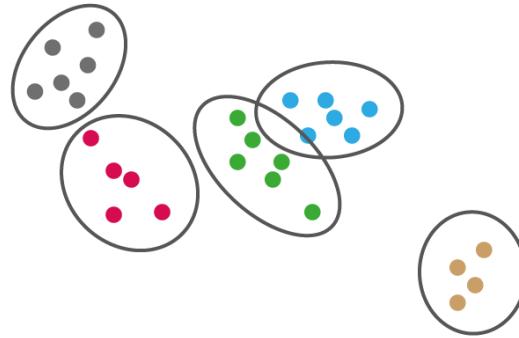


Рис. 1.4: Задача кластеризации

Примеры задач кластеризации:

- Сегментация пользователей (интернет-магазина или оператора связи)
- Поиск схожих пользователей в социальных сетях
- Поиск генов с похожими профилями экспрессии

1.3.2. Задача визуализации

Второй пример — задача визуализации: необходимо нарисовать многомерную (а конкретно, d -мерную) выборку так, чтобы изображение наглядно показывало структуру объектов.

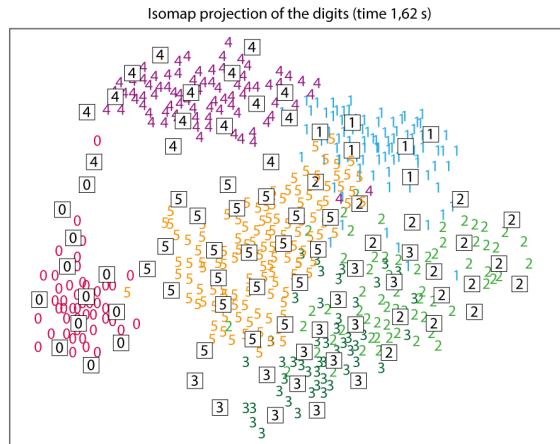


Рис. 1.5: Задача визуализации

Примером задачи визуализации является задача визуализации набора данных MNIST. Этот набор данных был получен в результате оцифровки рукописных начертаний цифр. Каждый скан цифры характеризуется

вектором признаков - яркостей отдельных пикселей. Необходимо таким образом отобразить этот набор данных на плоскость, чтобы разные цифры оказались в разных ее областях.

1.3.3. Поиск аномалий

Третий пример задачи обучения без учителя — поиск аномалий. Необходимо обнаружить, что данный объект не похож на все остальные, то есть является аномальным.

При обучении есть примеры только обычных, не аномальных, объектов. А примеров аномальных объектов либо нет вообще, либо настолько мало, что невозможно воспользоваться классическими методами обучения с учителем (методами бинарной классификации).

При этом задача очень важная. Например, к такому типу задач относится:

- Определение поломки в системах самолета (по показателям сотен датчиков)
- Определение поломки интернет—сайта
- Выявление проблем в модели машинного обучения.

Все упомянутые задачи не будут обсуждаться в рамках данного курса. Им будет посвящен следующий курс — «Поиск структуры в данных».

1.4. Признаки в машинном обучении

В этом разделе речь пойдет о признаках в машинном обучении. Существует несколько классов, или типов признаков. И у всех свои особенности — их нужно по-разному обрабатывать и по-разному учитывать в алгоритмах машинного обучения. В данном разделе будет обсуждаться используемая терминология, о самих же особенностях речь пойдет в следующих уроках.

Признаки описывают объект в доступной и понятной для компьютера форме. Множество значений j -го признака будет обозначаться D_j .

1.4.1. Бинарные признаки

Первый тип признаков — бинарные признаки. Они принимают два значения: $D_j = \{0, 1\}$. К таковым относятся:

- Выше ли доход клиента среднего дохода по городу?
- Цвет фрукта — зеленый?

Если ответ на вопрос да — признак полагается равным 1, если ответ на вопрос нет — то равным 0.

1.4.2. Вещественные признаки

Более сложный класс признаков — вещественные признаки. В этом случае $D_j = \mathbb{R}$. Примерами таких признаков являются:

- Возраст
- Площадь квартиры
- Количество звонков в call-центр

Множество значений последнего указанного признака, строго говоря, является множеством натуральных чисел \mathbb{N} , а не \mathbb{R} , но такие признаки тоже считают вещественными.

1.4.3. Категориальные признаки

Следующий класс признаков — категориальные признаки. В этом случае D_j — неупорядоченное множество. Отличительная особенность категориальных признаков — невозможность сравнения «больше-меньше» значений признака. К таковым признакам относятся:

- Цвет глаз
- Город
- Образование (В некоторых задачах может быть введен осмысленный порядок)

Категориальные признаки очень трудны в обращении — до сих пор появляются способы учета этих признаков в тех или иных методах машинного обучения.

1.4.4. Порядковые признаки

Частным случаем категориальных признаков являются порядковые признаки. В этом случае D_j — упорядоченное множество. Примеры:

- Роль в фильме (Первый план, второй план, массовка)
- Тип населенного пункта (упорядочены по населенности)
- Образование

Хотя и порядковые, и вещественные признаки упорядочены, они отличаются тем, что в случае порядковых признаков «расстояние» между двумя значениями признака не имеет смысла. Например, отличие значения 3 от значения 2 может быть не таким существенным, как отличие 1 от 0.

1.4.5. Множествозначные признаки

Множествозначный признак — это такой признак, значением которого на объекте является подмножество некоторого множества. Пример:

- Какие фильмы посмотрел пользователь
- Какие слова входят в текст

1.4.6. Распределение признака

Далее речь пойдет о проблемах, с которыми можно столкнуться при работе с признаками. Первая из них — существование выбросов. Выбросом называется такой объект, значение признака на котором отличается от значения признака на большинстве объектов.

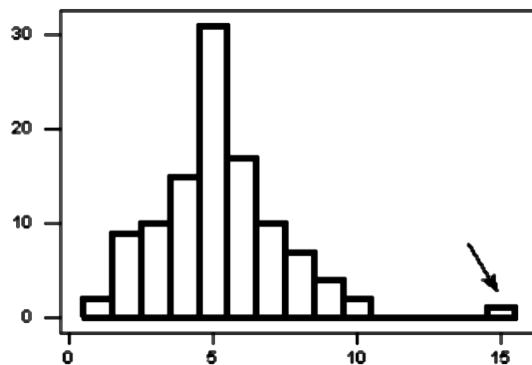


Рис. 1.6: Пример выброса

Наличие выбросов представляет сложность для алгоритмов машинного обучения, которые будут пытаться учесть и их тоже. Поскольку выбросы описываются совершенно другим законом, чем основное множество объектов, выбросы обычно исключают из данных, чтобы не мешать алгоритму машинного обучения искать закономерности в данных.

Проблема может быть и в том, как распределен признак. Не всегда признак имеет такое распределение, которое позволяет ответить на требуемый вопрос. Например, может быть слишком мало данных о клиентах из небольшого города, так как собрать достаточную статистику не представлялось возможным.

Урок 2

Линейные модели

2.1. Линейные модели в задачах регрессии

Данный урок будет посвящен линейным моделям. Речь пойдет о задачах классификации и регрессии, как их обучать, и с какими проблемами можно столкнуться при использовании этих моделей. В этом блоке мы обсудим, как выглядят линейные модели в задачах регрессии.

2.1.1. Повторение обозначений из прошлого урока

Для начала необходимо напомнить некоторые обозначения, которые были введены на прошлом уроке.

- \mathbb{X} — пространство объектов
- \mathbb{Y} — пространство ответов
- $x = (x^1, \dots, x^d)$ — признаковое описание объекта
- $X = (x_i, y_i)_{i=1}^\ell$ — обучающая выборка
- $a(x)$ — алгоритм, модель
- $Q(a, X)$ — функционал ошибки алгоритма a на выборке X
- Обучение: $a(x) = \operatorname{argmin}_{a \in \mathbb{A}} Q(a, X)$

Напомним, что в задаче регрессии пространство ответов $\mathbb{Y} = \mathbb{R}$. Чтобы научиться решать задачу регрессии, необходимо задать:

- **Функционал ошибки Q :** способ измерения того, хорошо или плохо работает алгоритм на конкретной выборке.
- **Семейство алгоритмов \mathbb{A} :** как выглядит множество алгоритмов, из которых выбирается лучший.
- **Метод обучения:** как именно выбирается лучший алгоритм из семейства алгоритмов.

2.1.2. Пример задачи регрессии: предсказание прибыли магазина

Пусть известен один признак — прибыль магазина в прошлом месяце, а предсказать необходимо прибыль магазина в следующем. Поскольку прибыль — вещественная переменная, здесь идет речь о задаче регрессии.

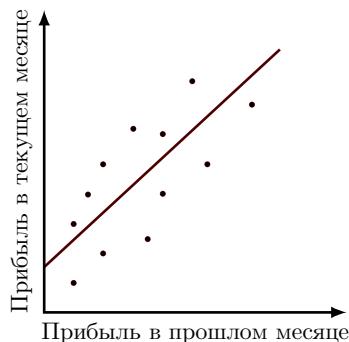


Рис. 2.1: Точки обучающей выборки.

По этому графику можно сделать вывод о существовании зависимости между прибылью в следующем и прошлом месяцах. Если предположить, что зависимость приблизительно линейная, ее можно представить в виде прямой на этом графике. По этой прямой и можно будет предсказывать прибыль в следующем месяце, если известна прибыль в прошлом.

В целом такая модель угадывает тенденцию, то есть описывает зависимость между ответом и признаком. При этом, разумеется, она делает это не идеально, с некоторой ошибкой. Истинный ответ на каждом объекте несколько отклоняется от прогноза.

Один признак — это не очень серьезно. Гораздо сложнее и интереснее работать с многомерными выборками, которые описываются большим количеством признаков. В этом случае нарисовать выборку и понять, подходит или нет линейная модель, нельзя. Можно лишь оценить ее качество и по нему уже понять, подходит ли эта модель.

Следует отметить, что вообще нельзя придумать модель, которая идеально описывает ваши данные, то есть идеально описывает, как порождается ответ по признакам.

2.1.3. Описание линейной модели

Далее обсудим, как выглядит семейство алгоритмов в случае с линейными моделями. Линейный алгоритм в задачах регрессии выглядит следующим образом:

$$a(x) = w_0 + \sum_{j=1}^d w_j x^j,$$

где w_0 — свободный коэффициент, x^j — признаки, а w_j — их веса.

Если добавить $(d+1)$ -й признак, который на каждом объекте принимает значение 1, линейный алгоритм можно будет записать в более компактной форме

$$a(x) = \sum_{j=1}^{d+1} w_j x^j = \langle w, x \rangle,$$

где используется обозначение $\langle w, x \rangle$ для скалярного произведения двух векторов.

В качестве меры ошибки не может быть выбрано отклонение от прогноза $Q(a, y) = a(x) - y$, так как в этом случае минимум функционала не будет достигаться при правильном ответе $a(x) = y$. Самый простой способ — считать модуль отклонения:

$$|a(x) - y|.$$

Но функция модуля не является гладкой функцией, и для оптимизации такого функционала неудобно использовать градиентные методы. Поэтому в качестве меры ошибки часто выбирается квадрат отклонения:

$$(a(x) - y)^2.$$

Функционал ошибки, именуемый среднеквадратичной ошибкой алгоритма, задается следующим образом:

$$Q(a, x) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2$$

В случае линейной модели его можно переписать в виде функции (поскольку теперь Q зависит от вектора, а не от функции) ошибок:

$$Q(w, x) = \frac{1}{\ell} \sum_{i=1}^{\ell} (\langle w, x_i \rangle - y_i)^2.$$

2.2. Обучение модели линейной регрессии

В этом блоке речь пойдет о том, как обучать модель линейной регрессии, то есть как настраивать ее параметры. В прошлый раз было введено следующее выражение для качества линейной модели на обучающей выборке:

$$Q(w, x) = \frac{1}{\ell} \sum_{i=1}^{\ell} (\langle w, x_i \rangle - y_i)^2 \rightarrow \min_w.$$

Следует напомнить, что в число признаков входит также постоянный признак, равный 1 для всех объектов, что позволяет исключить постоянную составляющую в последнем соотношении.

2.2.1. Переход к матричной форме записи

Прежде, чем будет рассмотрена задача оптимизации этой функции, имеет смысл переписать используемые соотношения в матричной форме. Матрица «объекты–признаки» X составлена из признаковых описаний всех объектов из обучающей выборки:

$$X = \begin{pmatrix} x_{11} & \dots & x_{1d} \\ \dots & \dots & \dots \\ x_{\ell 1} & \dots & x_{\ell d} \end{pmatrix}$$

Таким образом, в ij элементе матрицы X записано значение j -го признака на i объекте обучающей выборки. Также понадобится вектор ответов y , который составлен из истинных ответов для всех объектов:

$$y = \begin{pmatrix} y_1 \\ \dots \\ y_\ell \end{pmatrix}.$$

В этом случае среднеквадратичная ошибка может быть переписана в матричном виде:

$$Q(w, X) = \frac{1}{\ell} \|Xw - y\|^2 \rightarrow \min_w.$$

Эта формула пригодится, в частности, при реализации линейной регрессии на компьютере.

2.2.2. Аналитический метод решения

Можно найти аналитическое решение задачи минимизации:

$$w_* = (X^T X)^{-1} X^T y.$$

Основные сложности при нахождении решения таким способом:

- Для нахождения решения необходимо вычислять обратную матрицу. Операция обращения матрицы требует, в случае d признаков, выполнение порядка d^3 операции, и является вычислительно сложной уже в задачах с десятком признаков.
- Численный способ нахождения обратной матрицы не может быть применен в некоторых случаях (когда матрица плохо обусловлена).

2.2.3. Оптимизационный подход к решению

Другой, несколько более удобный, способ найти решение — использовать численные методы оптимизации.

Несложно показать, что среднеквадратическая ошибка — это выпуклая и гладкая функция. Выпуклость гарантирует существование лишь одного минимума, а гладкость — существование вектора градиента в каждой точке. Это позволяет использовать метод градиентного спуска.

При использовании метода градиентного спуска необходимо указать начальное приближение. Есть много подходов к тому, как это сделать, в том числе инициализировать случайными числами (не очень большими). Самый простой способ это сделать — инициализировать значения всех весов равными нулю:

$$w^0 = 0.$$

На каждой следующей итерации, $t = 1, 2, 3, \dots$, из приближения, полученного в предыдущей итерации w^{t-1} , вычитается вектор градиента в соответствующей точке w^{t-1} , умноженный на некоторый коэффициент η_t , называемый шагом:

$$w^t = w^{t-1} - \eta_t \nabla Q(w^{t-1}, X).$$

Остановить итерации следует, когда наступает сходимость. Сходимость можно определять по-разному. В данном случае разумно определить сходимость следующим образом: итерации следует завершить, если разница двух последовательных приближений не слишком велика:

$$\|w^t - w^{t-1}\| < \varepsilon.$$

Подробно метод градиентного спуска обсуждался в прошлом курсе этой специализации.

2.3. Градиентный спуск для линейной регрессии

2.3.1. Случай парной регрессии

В случае парной регрессии признак всего один, а линейная модель выглядит следующим образом:

$$a(x) = w_1 x + w_0,$$

где w_1 и w_0 — два параметра.

Среднеквадратичная ошибка принимает вид:

$$Q(w_0, w_1, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (w_1 x_i + w_0 - y_i)^2.$$

Для нахождения оптимальных параметров будет применяться метод градиентного спуска, про который уже было сказано ранее. Чтобы это сделать, необходимо сначала вычислить частные производные ошибки:

$$\frac{\partial Q}{\partial w_1} = \frac{2}{\ell} \sum_{i=1}^{\ell} (w_1 x_i + w_0 - y_i) x_i, \quad \frac{\partial Q}{\partial w_0} = \frac{2}{\ell} \sum_{i=1}^{\ell} (w_1 x_i + w_0 - y_i).$$

2.3.2. Демонстрация градиентного спуска в случае парной регрессии

Следующие два графика демонстрируют применение метода градиентного спуска в случае парной регрессии. Справа изображены точки выборки, а слева — пространство параметров. Точка в этом пространстве обозначает конкретную модель.

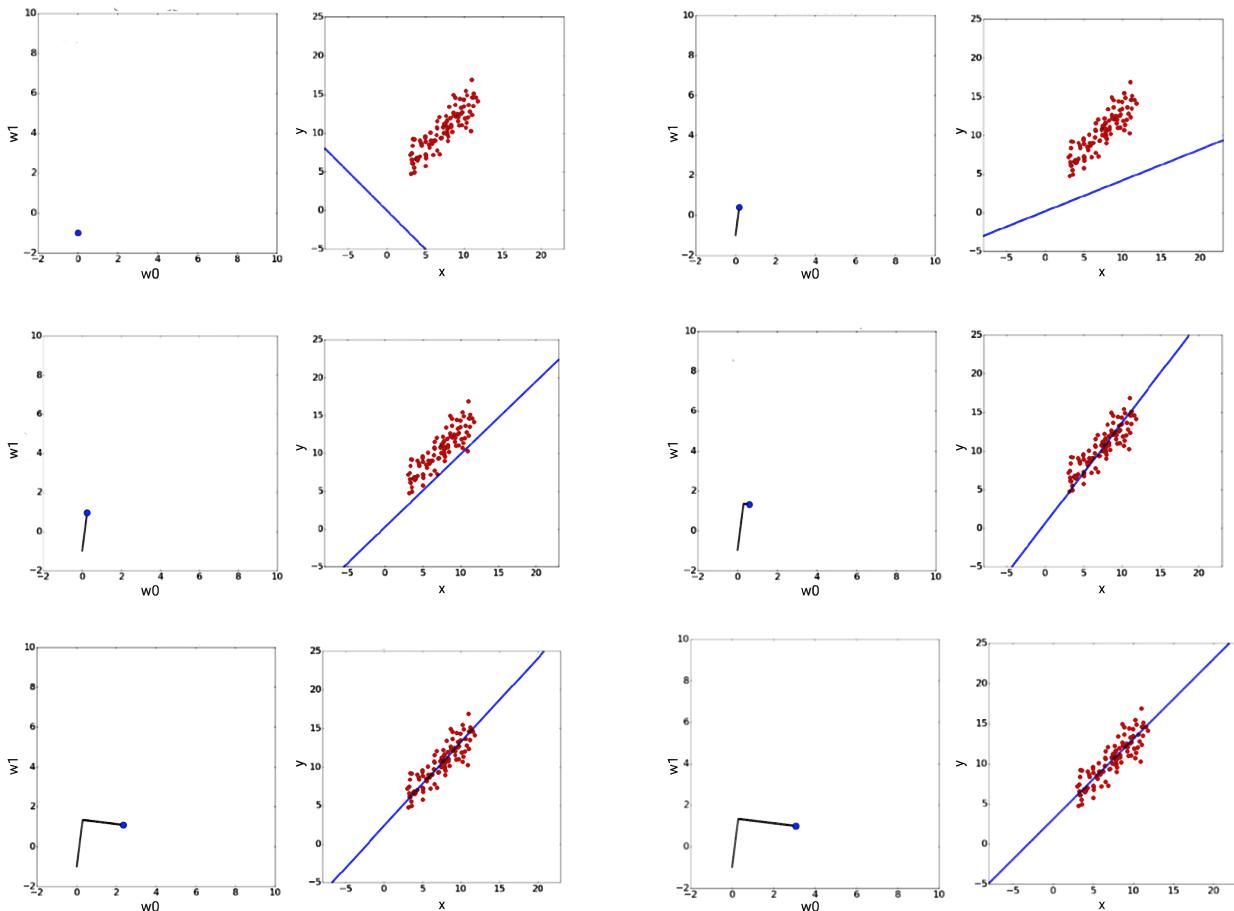


Рис. 2.2: Демонстрация метода градиентного спуска.

График зависимости функции ошибки от числа произведенных операции выглядит следующим образом:

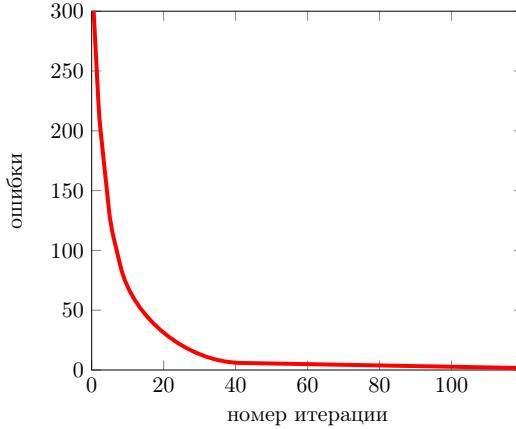


Рис. 2.3: Ошибка в зависимости от номера итерации

2.3.3. Выбор размера шага в методе градиентного спуска

Очень важно при использовании метода градиентного спуска правильно подбирать шаг. Каких-либо конкретных правил подбора шага не существует, выбор шага — это искусство, но существует несколько полезных закономерностей.

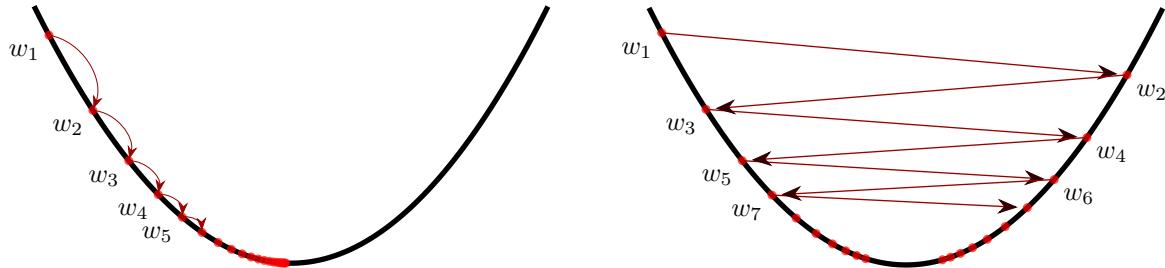


Рис. 2.4: Случай маленького и большого шага

Если длина шага слишком мала, то метод будет неспешно, но верно шагать в сторону минимума. Если же взять размер шага очень большим, появляется риск, что метод будет перепрыгивать через минимум. Более того, есть риск того, что градиентный спуск не сойдется.

Имеет смысл использовать переменный размер шага: сначала, когда точка минимума находится еще далеко, двигаться быстро, а позже, спустя некоторое количество итераций — делать более аккуратные шаги. Один из способов задать размер шага следующий:

$$\eta_t = \frac{k}{t},$$

где k — константа, которую необходимо подобрать, а t — номер шага.

2.3.4. Случай многомерной линейной регрессии

В случае многомерной линейной регрессии используется тот же самый подход — необходимо решать задачу минимизации:

$$Q(w, X) = \frac{1}{\ell} \|Xw - y\|^2 \rightarrow \min_w,$$

где $\|x\|$ — норма вектора x . Формула для вычисления градиента принимает следующий вид:

$$\nabla_w Q(w, X) = \frac{2}{\ell} X^T (Xw - y)$$

Стоит отметить, что вектор $Xw - y$, который присутствует в данном выражении, представляет собой вектор ошибок.

2.4. Стохастический градиентный спуск

В этом блоке речь пойдет о стохастическом градиентном спуске, который особенно хорошо подходит для обучения линейных моделей.

2.4.1. Недостатки обычного метода градиентного спуска

В обычном методе градиентного спуска на каждом шаге итерации следующее приближение получается из предыдущего вычитанием вектора градиента, умноженного на шаг η_t :

$$w^t = w^{t-1} - \eta_t \nabla Q(w^{t-1}, X).$$

При этом выражение для градиента в матричной форме имеет вид:

$$\nabla_w Q(w, X) = \frac{2}{\ell} X^T (Xw - y)$$

Выражение для j -ой компоненты градиента, таким образом, содержит суммирование по всем объектам обучающей выборки:

$$\frac{\partial Q}{\partial w_j} = \frac{2}{\ell} \sum_{i=1}^{\ell} x_i^j (\langle w, x_i \rangle - y_i).$$

В этом и состоит основной недостаток метода градиентного спуска — в случае большой выборки даже одна итерация метода градиентного спуска будет производиться долго.

2.4.2. Стохастический градиентный спуск

Идея стохастического градиентного спуска основана на том, что в сумме в выражении для j -компоненты градиента i -ое слагаемое указывает то, как нужно поменять вес w_j , чтобы качество увеличилось для i -го объекта выборки. Вся сумма при этом задает, как нужно изменить этот вес, чтобы повысить качество для всех объектов выборки. В стохастическом методе градиентного спуска градиент функции качества вычисляется только на одном случайно выбранном объекте обучающей выборки. Это позволяет обойти вышеупомянутый недостаток обычного градиентного спуска.

Таким образом, алгоритм стохастического градиентного спуска следующий. Сначала выбирается начальное приближение:

$$w^0 = 0$$

Далее последовательно вычисляются итерации w^t : сначала случайным образом выбирается объект x_i из обучающей выборки X и вычисляется вектор градиента функции качества на этом объекте, а следующее приближение получается из предыдущего вычитанием умноженного на шаг η_t полученного вектора:

$$w^t = w^{t-1} - \eta_t \nabla Q(w^{t-1}, \{x_i\}).$$

Итерации прекращаются при достижении определенного условия, например:

$$\|w^t - w^{t-1}\| < \varepsilon.$$

2.4.3. Сходимость стохастического градиентного спуска

Показательно посмотреть на графики сходимости градиентного спуска и стохастического градиентного спуска. В обычном градиентном спуске на каждом шаге уменьшается суммарная ошибка на всех элементах обучающей выборки. График в таком случае обычно получается монотонным.

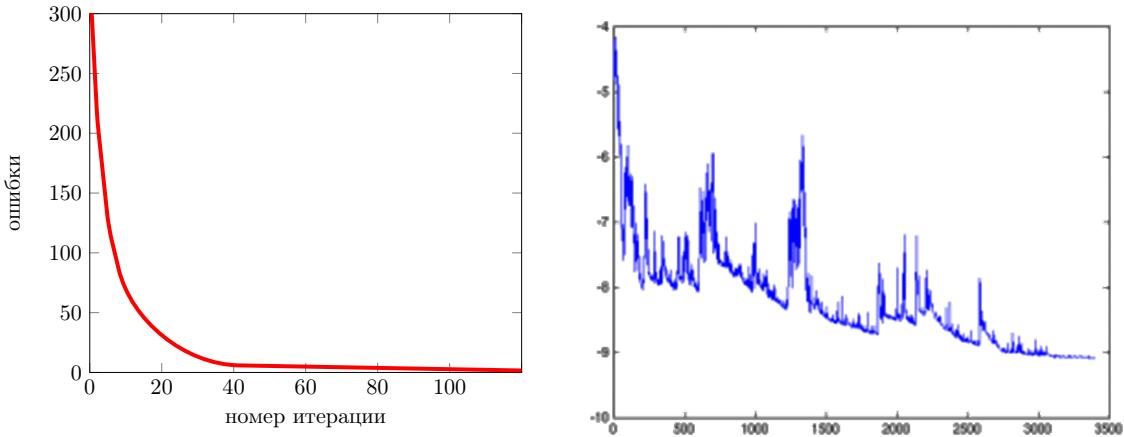


Рис. 2.5: Ошибка в зависимости от номера итерации

Напротив, в стохастическом методе весовые коэффициенты меняются таким образом, чтобы максимально уменьшить ошибку для одного случайно выбранного объекта. Это приводит к тому, что график выглядит пилообразным, то есть на каждой конкретной итерации полная ошибка может как увеличиваться, так и уменьшаться. Но в итоге с ростом номера итерации значение функции уменьшается.

2.4.4. Особенности стохастического градиентного спуска

Стохастический градиентный спуск (SGD) обладает целым рядом преимуществ. Во-первых, каждый шаг выполняется существенно быстрее шага обычного градиентного метода, а также не требуется постоянно хранить всю обучающую выборку в памяти. Это позволяет использовать для обучения выборки настолько большие, что они не помещаются в память компьютера. Стохастический градиентный спуск также можно использовать для онлайн-обучения, то есть в ситуации, когда на каждом шаге алгоритм получает только один объект и должен учесть его для коррекции модели.

2.5. Линейная классификация

2.5.1. Задача бинарной классификации

В этом разделе рассматривается задача линейной классификации, то есть применение линейных моделей к задаче классификации. Речь пойдет о самом простом виде классификации — бинарной классификации. В случае бинарной классификации множество возможных значений ответов состоит из двух элементов:

$$\mathbb{Y} = \{-1, +1\}.$$

Как уже было сказано, чтобы работать с той или иной моделью нужно:

- Выбрать функционал (функцию) ошибки, то есть задать способ определения качества работы того или иного алгоритма на обучающей выборке.
- Построить семейство алгоритмов, то есть множество алгоритмов, из которого потом будет выбираться наилучший с точки зрения определенного функционала ошибки.
- Ввести метод обучения, то есть определить способ выбора лучшего алгоритма из семейства.

2.5.2. Линейный классификатор

Ранее была рассмотрена задача линейной регрессии. В этом случае алгоритм представлял собой линейную комбинацию признаков с некоторыми весами и свободным коэффициентом.

Линейные классификаторы устроены похожим образом, но они должны возвращать бинарные значения, а следовательно требуется также брать знак от получившегося выражения:

$$a(x) = \text{sign} \left(w_0 + \sum_{j=1}^d w_j x^j \right).$$

Как и раньше, добавлением еще одного постоянного для всех объектов признака можно привести формулу к более однородному виду:

$$a(x) = \operatorname{sign} \sum_{j=1}^{d+1} w_j x^j = \operatorname{sign} \langle \vec{w}, x \rangle$$

2.5.3. Геометрический смысл линейного классификатора

Выражение $\langle \vec{w}, x \rangle = 0$ является уравнением некоторой плоскости в пространстве признаков.

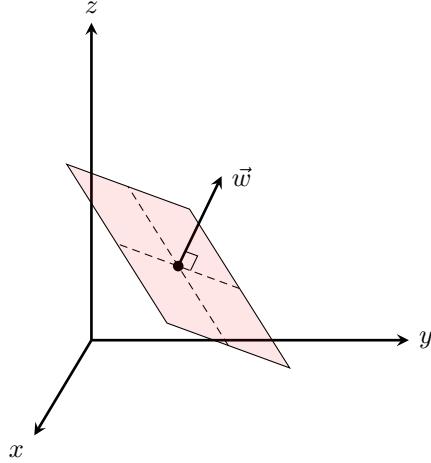


Рис. 2.6: Геометрический смысл линейного классификатора

При этом для точек по одну сторону от этой плоскости скалярное произведение $\langle \vec{w}, x \rangle$ будет положительным, а с другой — отрицательным.

Таким образом, линейный классификатор проводит плоскость в пространстве признаков и относит объекты по разные стороны от нее к разным классам.

Согласно геометрическому смыслу скалярного произведения, расстояние от конкретного объекта, который имеет признаковое описание x , до гиперплоскости $\langle \vec{w}, x \rangle = 0$ равно $\frac{|\langle \vec{w}, x \rangle|}{\|\vec{w}\|}$. С этим связано такое важное понятие в задачах линейной классификации как понятие отступа:

$$M_i = y_i \langle \vec{w}, x_i \rangle.$$

Отступ является величиной, определяющей корректность ответа. Если отступ больше нуля $M_i > 0$, то классификатор дает верный ответ для i -го объекта, в ином случае — ошибается.

Причем чем дальше отступ от нуля, тем больше уверенность как в правильном ответе, так и в том, что алгоритм ошибается. Если отступ для некоторого объекта отрицательный и большой по модулю, это значит, что алгоритм неправильно описывает данные: либо этот объект является выбросом, либо алгоритм не пригоден для решения данной задачи.

2.6. Функции потерь в задачах классификации

2.6.1. Пороговая функция потерь

В случае линейной классификации естественный способ определить качество того или иного алгоритма — вычислить для объектов обучающей выборки долю неправильных ответов:

$$Q(a, x) = \frac{1}{\ell} \sum_{i=1}^{\ell} [a(x_i) \neq y_i]$$

С помощью введенного ранее понятия отступа можно переписать это выражение для случая линейной классификации в следующем виде:

$$Q(a, x) = \frac{1}{\ell} \sum_{i=1}^{\ell} [y_i \langle w, x_i \rangle < 0] = \frac{1}{\ell} \sum_{i=1}^{\ell} [M_i < 0]$$

Функция, стоящая под знаком суммы, называется функцией потерь. В данном случае это пороговая функция потерь, график которой в зависимости от отступа выглядит следующим образом:

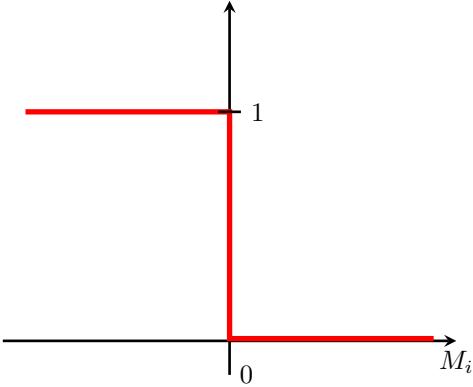


Рис. 2.7: График пороговой функции потерь

Такая функция является разрывной в точке 0, что делает невозможным применение метода градиентного спуска. Можно, конечно, использовать методы негладкой оптимизации, о которых шла речь в прошлом курсе, но они сложны в реализации.

2.6.2. Оценка функции потерь

Используя любую гладкую оценку пороговой функции:

$$[M_i < 0] \leq \tilde{L}(M_i)$$

можно построить оценку $\tilde{Q}(a, X)$ для функционала ошибки $Q(a, X)$:

$$Q(a, X) \leq \tilde{Q}(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} \tilde{L}(M_i).$$

В этом случае минимизировать нужно будет не долю неправильных ответов, а некоторую другую функцию, которая является оценкой сверху:

$$Q(a, X) \leq \tilde{Q}(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} \tilde{L}(M_i) \rightarrow \min_a.$$

Здесь используется предположение, что в точке минимума этой верхней оценки число ошибок также будет минимально. Строго говоря, это не всегда так.

2.6.3. Примеры оценок функции потерь

Примерами таких оценок функции потерь являются:

- Логистическая функция потерь (используется в логистической регрессии, о которой пойдет речь позже в данном курсе):

$$\tilde{L}(M) = \log_2 (\exp(-M))$$

- Экспоненциальная функция потерь:

$$\tilde{L}(M) = \exp(-M)$$

- Кусочно-линейная функция потерь (используется в методе опорных векторов):

$$\tilde{L}(M) = \max(0, 1 - M)$$

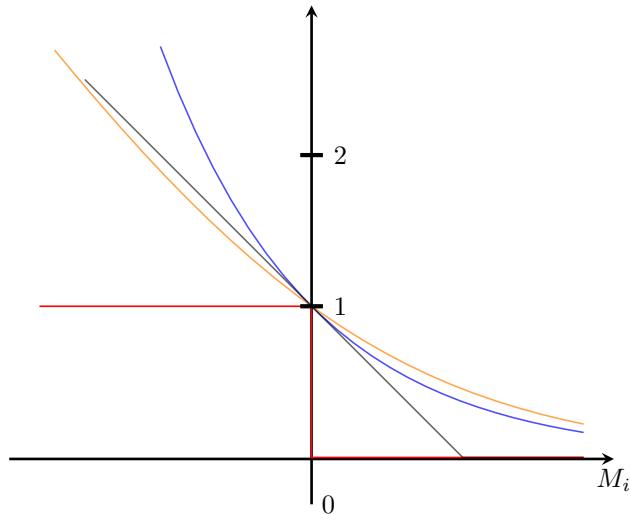


Рис. 2.8: Графики различных функций потерь: пороговая (красная линия), экспоненциальная (синяя), логистическая (оранжевая) и кусочно-линейная (серая).

2.6.4. Логистическая функция потерь

В случае логистической функции потерь функционал ошибки имеет вид:

$$\tilde{Q}(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} \ln (\exp(-M_i)) = \frac{1}{\ell} \sum_{i=1}^{\ell} \ln (\exp(-y_i \langle w, x_i \rangle)).$$

Получившееся выражение является гладким, а, следовательно, можно использовать, например, метод градиентного спуска.

Следует обратить внимание, что в случае, если число ошибок стало равно нулю, все равно в ходе обучения алгоритма линейной классификации будут увеличиваться отступы, то есть будет увеличиваться уверенность в полученных результатах.

Урок 3

Проблема переобучения и борьба с ней

3.1. Проблема переобучения

3.1.1. Пример: проблема переобучения в задачах классификации

Допустим при решении задачи классификации был построен некоторый алгоритм, например линейный классификатор, причем доля ошибок на объектах из обучающей выборки была равна 0.2, и такая доля ошибок является допустимой.

Но поскольку алгоритм не обладает обобщающей способностью, нет никаких гарантий, что такая же доля ошибок будет для новой выборки. Вполне может возникнуть ситуация, что для новой выборки ошибка станет равной 0.9. Это значит, что алгоритм не смог обобщить обучающую выборку, не смог извлечь из нее закономерности и применить их для классификации новых объектов. При этом алгоритм как-то смог подогнаться под обучающую выборку и показал хорошие результаты при обучении без извлечения истинной закономерности. В этом и состоит проблема переобучения.

3.1.2. Пример: проблема переобучения в задачах линейной регрессии

Глубже понять проблему переобучения можно на данном примере. На следующем графике изображена истинная зависимость и объекты обучающей выборки:

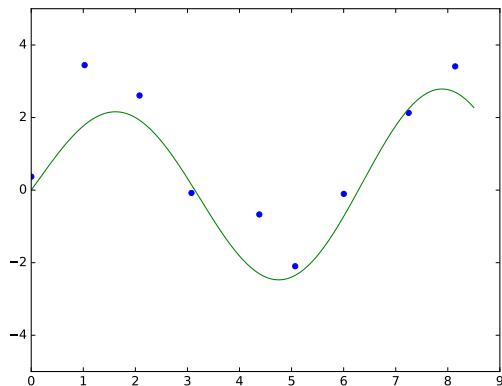


Рис. 3.1: Истинная зависимость (зеленая линия) и элементы обучающей выборки (изображены синими точками).

Видно, что истинная зависимость является нелинейной и имеет два экстремума.

В модели $a(x) = w_0$, после того, как она будет настроена под данные, на графике получается некоторая горизонтальная кривая, которая довольно плохо обобщает информацию об объектах из выборки.

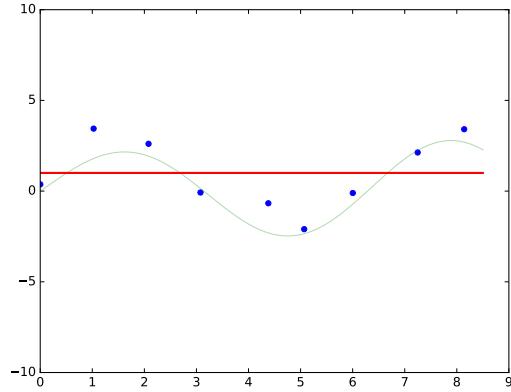


Рис. 3.2: Модель $a(x) = w_0$.

Имеет место недообучение. Хороший алгоритм не был построен, поскольку семейство алгоритмов слишком мало и с его помощью невозможно уловить закономерность.

В линейной регрессии используется семейство алгоритмов $a(x) = w_0 + w_1x$.

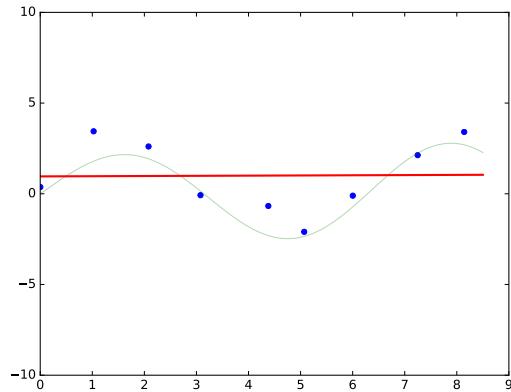


Рис. 3.3: Модель $a(x) = w_0 + w_1x$.

В этом случае также будет иметь место недообучение. Получилось лучше, но прямая тоже плохо описывает данные.

Если семейство алгоритмов — множество многочленов 4-ей степени:

$$a(x) = w_0 + w_1x + w_2x^2 + \dots + w_4x^4,$$

то после обучения получившаяся кривая будет достаточно хорошо описывать и обучающую выборку, и истинную зависимость.

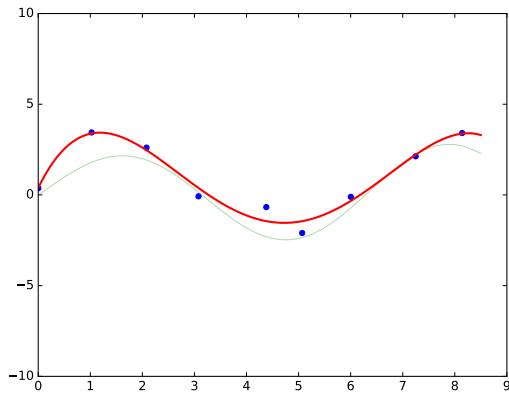


Рис. 3.4: Модель $a(x) = w_0 + w_1x + w_2x^2 + \dots + w_4x^4$.

В таком случае качество алгоритма хорошее, но нет идеального совпадения. Встает вопрос, а можно ли добиться совпадения увеличением сложности алгоритма.

При использовании многочленов 9-ой степени уже имеет место переобучение.

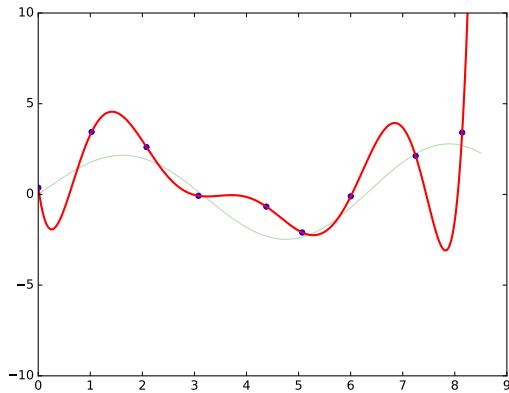


Рис. 3.5: Модель $a(x) = w_0 + w_1x + w_2x^2 + \dots + w_9x^9$.

Восстановленная зависимость дает идеальные ответы на всех объектах обучающей выборки, но при этом в любой другой точке сильно отличается от истинной зависимости. Такая ситуация называется переобучением. Алгоритм слишком сильно подогнался под обучающую выборку ценой того, что он будет давать плохие ответы на новых точках.

3.1.3. Недообучение и переобучение

Таким образом, недообучение — ситуация, когда алгоритм плохо описывает и обучающую выборку, и новые данные. В этом случае алгоритм необходимо усложнять.

В случае переобучения, данные из обучающей выборки будут описываться хорошо, а новые данные плохо. Выявить переобучение, используя только обучающую выборку, невозможно, поскольку и хорошо обученный, и переобученный алгоритмы будут хорошо ее описывать. Необходимо использовать дополнительные данные.

Существуют несколько подходов к выявлению переобучения:

- Отложенная выборка. Часть данных из обучающей выборки не участвуют в обучении, чтобы позже проверять на ней обученный алгоритм.
- Кросс-валидация, несколько усложненный метод отложенной выборки. (Об этом способе речь пойдет позже.)

- Использовать меры сложности модели. Об этом пойдет речь далее.

3.2. Регуляризация

В этом разделе речь пойдет о регуляризации — способе борьбы с переобучением в линейных моделях.

3.2.1. «Симптомы» переобучения. Мультиколлинеарность.

Мерой сложности, то есть «симптомом» переобученности модели, являются большие веса при признаках. Например, в предыдущем разделе при обучении модели

$$a(x) = w_0 + w_1x + w_2x^2 + \dots + w_9x^9$$

веса оказывались огромными:

$$a(x) = 0.5 + 12458922x + 43983740x^2 + \dots + 2740x^9.$$

Другая ситуация, в которой можно встретиться с переобучением — мультиколлинеарность. Так называется проблема, при которой признаки в выборке являются линейно зависимыми. Другими словами, существуют коэффициенты $\alpha_1, \dots, \alpha_d$ такие, что для любого объекта x_i из выборки выполняется:

$$\alpha_1x_i^1 + \dots + \alpha_dx_i^d = 0.$$

Более компактно последнее выражение можно переписать в виде:

$$\langle \alpha, x_i \rangle = 0.$$

Допустим, было найдено решение задачи оптимизации:

$$w_* = \operatorname{argmin}_w \frac{1}{\ell} \sum_{i=1}^{\ell} (\langle w, x_i \rangle - y_i)^2$$

Другой вектор весов, полученный сдвигом в направлении вектора α :

$$w_1 = w_* + t\alpha,$$

так как для элементов x выборки выполняется:

$$\langle w_* + t\alpha, x \rangle = \langle w_*, x \rangle + t\langle \alpha, x \rangle = \langle w_*, x \rangle,$$

также будет являться решением задачи оптимизации. Другими словами, он будет также хорошо описывать данные в выборке, как и исходный алгоритм. Фактически, решениями задачи оптимизации являются бесконечное множество алгоритмов, но многие из них имеют большие веса, и далеко не все обладают хорошей обобщающей способностью. Поэтому здесь тоже легко столкнуться с переобучением.

3.2.2. Регуляризация

Выше было продемонстрировано, что если веса в линейной модели большие, существует высокий риск переобучения. Чтобы бороться с этим, минимизируется уже не выражение для функционала ошибки $Q(a, X)$, а новый функционал, получаемый прибавлением регуляризатора. Самый простой регуляризатор — квадратичный регуляризатор:

$$\|w\|^2 = \sum_{j=1}^d w_j^2.$$

В этом случае имеет место следующая задача оптимизации:

$$Q(w, X) + \lambda \|w\|^2 \rightarrow \min_w.$$

Таким образом, при обучении будет учитываться также то, что не следует слишком сильно увеличивать веса признаков.

3.2.3. Коэффициент регуляризации

Введенный выше коэффициент λ , который стоит перед регуляризатором, называется коэффициентом регуляризации. Чем больше λ , тем ниже сложность модели. Например, при очень больших его значениях оптимально просто занулить все веса. В то же время при слишком низких значениях λ высок риск переобучения, то есть модель становится слишком сложной.

Поэтому нужно найти некоторое оптимальное значение λ , достаточно большое, чтобы не допустить переобучения, и не очень большое, чтобы уловить закономерности в данных. Обычно λ подбирается на кроссвалидации, о которой пойдет речь в следующем разделе.

3.2.4. Смысл регуляризации

Чтобы понять смысл регуляризации, вместо задачи оптимизации с квадратичным оптимизатором нагляднее рассмотреть задачу условной оптимизации:

$$\begin{cases} Q(w, X) \rightarrow \min_w \\ \|w\|^2 \leq C \end{cases}$$

Добавление регуляризатора вводит требование, чтобы решение задачи минимизации искалось в некоторой круглой области с центром в нуле.

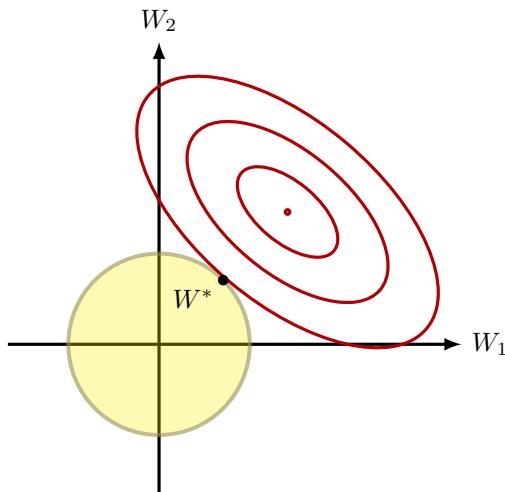


Рис. 3.6: Геометрический смысл условной регуляризации. Красная точка — настоящий оптимум функции, красные линии — линии уровня функции, черная точка — оптимум функции при введенном ограничении.

Таким образом, решение задачи с регуляризатором не будет характеризоваться слишком большими значениями весовых коэффициентов.

3.2.5. Виды регуляризаторов

Рассмотренный выше квадратичный регуляризатор (L_2 -регуляризатор) является гладким и выпуклым, что позволяет использовать градиентный спуск.

Также существует L_1 -регуляризатор:

$$\|w\|_1 = \sum_{j=1}^d |w_j|,$$

который представляет собой L_1 -норму вектора весов. Он уже не является гладким, а также обладает интересным свойством. Если применять такой регуляризатор, некоторые веса оказываются равными нулю. Другими словами, такой регуляризатор производит отбор признаков и позволяет использовать в модели не все признаки, а только самые важные из них.

3.3. Оценка качества алгоритмов. Кросс-валидация

В этом разделе речь пойдет об оценке качества алгоритмов и о том, как понять, как поведет себя алгоритм на новых данных.

3.3.1. Выявление переобучения

Уже было сказано, что переобучение сложно выявить, используя только обучающую выборку: и хороший, и переобученный алгоритмы будут показывать хорошее качество на объектах обучающей выборки. Рассмотренные в предыдущем разделе меры переобученности (значения регуляризаторов), безусловно, можно применять, но они не дают ответа на вопрос, насколько хорошо алгоритм поведет себя на новых данных, то есть какая у него будет доля ошибок на новых данных.

3.3.2. Отложенная выборка

Самый простой способ оценить качество алгоритма — использование отложенной выборки. В этом случае следует разбить выборку на две части: первая из двух частей будет использоваться для обучения алгоритма, а вторая, тестовая выборка, — для оценки его качества, в том числе для нахождения доли ошибок в задаче классификации, MSE (среднеквадратичной ошибки) в задаче регрессии и других мер качества в зависимости от специфики задачи.

Естественный вопрос — о том, в какой пропорции производить разбиение. Если взять тестовую выборку слишком маленькой, оценка качества будет ненадежной, хотя обучающая выборка будет почти совпадать с полной выборкой. В противоположном случае, если отложенная часть будет большой, оценка качества будет надежной, но низкое качество алгоритма может свидетельствовать о недостаточном объеме первой, обучающей, части выборки. Обычно выборку разбивают в соотношениях 70/30, 80/20 или 0.632/0.368.

Преимуществом отложенной выборки является то, что обучать алгоритм приходится всего лишь один раз, но при этом результат сильно зависит от того, как было произведено разбиение.

Например, оценивается стоимость жилья по некоторым признакам. И есть особая категория жилья, например двухэтажные квартиры. И если окажется, что все двухэтажные квартиры, которых немного, попали в отложенную выборку, то после обучения алгоритм будет давать на них очень плохое качество, поскольку в обучающей выборке таких объектов не было.

Чтобы решить эту проблему, можно использовать следующий подход: построить n различных разбиений выборки на 2 части, для каждого разбиения найти оценку качества, а в качестве итоговой оценки качества работы алгоритма использовать усредненное по всем разбиениям значение. Но и в данном случае, поскольку разбиения строятся случайно, нет никаких гарантий, что особый объект хотя бы раз попадет на обучение.

3.3.3. Кросс-валидация

Более системный подход — кросс-валидация. В этом случае выборка делится на k блоков примерно одинакового размера. Далее по очереди каждый из этих блоков используется в качестве тестового, а все остальные — в качестве обучающей выборки.

После того, как каждый блок побывает в качестве тестового, будут получены k показателей качества. В результате усреднения получается оценка качества по кросс-валидации.

При этом встает вопрос, какое число блоков использовать. Если блоков мало, получаются надежные, но смещенные оценки. В случае большого числа блоков оценки, наоборот, получаются ненадежными (большой разброс оценок), но несмещенными.

Нет конкретных рекомендаций относительно выбора k . Обычно выбирают $k = 3, 5, 10$. Чем больше k , тем больше раз приходится обучать алгоритм. Поэтому на больших выборках следует выбирать небольшие значения k , так как даже при удалении 1/3 выборки (а она большая) оставшихся данных будет достаточно для обучения.

3.3.4. Совет: перемешивайте данные в выборке

Часто данные в файле записаны в отсортированном виде по какого-нибудь признаку. Поэтому всегда следует перемешивать выборку прежде, чем производить кросс-валидацию. В ином случае алгоритм будет показывать плохое качество и причина этого будет не так очевидна.

При этом есть задачи, в которых выборку нельзя перемешивать. Это задачи предсказания будущего, например предсказание погоды на следующий день. В этом случае нужно особо следить за тем, как происходит деление выборки.

3.4. Выбор гиперпараметров и сравнение алгоритмов

В этом разделе речь пойдет о выборе гиперпараметров и сравнении различных алгоритмов с помощью изученных ранее схем.

3.4.1. Гиперпараметры

Гиперпараметрами называются такие параметры алгоритмов, которые не могут быть получены из обучающей выборки при обучении, поэтому их надо подбирать путем многократного обучения алгоритма. Примерами гиперпараметров являются:

- Параметр регуляризации λ (при использовании регуляризатора)
- Степень полинома в задаче регрессии с семейством алгоритмов, заданным множеством полиномов определенной степени.

3.4.2. Сравнение разных алгоритмов

Более общая задача — сравнение разных алгоритмов:

- обученных с разными значениями гиперпараметров;
- использующих различный способ регуляризации;
- настроенных с использованием разного функционала ошибки, например среднеквадратичной ошибки и средней абсолютной ошибки;
- которые принадлежат разным классам алгоритмов.

При сравнении алгоритмов можно использовать как отложенную выборку, так и кросс-валидацию, но при этом следует соблюдать осторожность.

Действительно, пусть 1000 алгоритмов сравниваются по качеству на отложенной выборке. Каждый из 1000 алгоритмов, обученных на обучающей выборке, тестируется на отложенной, и в результате выбирается лучший. Фактически на этом шаге отложенная выборка также становится своего рода обучающей, и возникает проблема переобучения: из большого числа алгоритмов выбирается тот, который лучше всего ведет себя на отложенной выборке, лучше подогнан под нее.

3.4.3. Улучшенная схема сравнения алгоритмов

Чтобы бороться с этим, следует использовать несколько усовершенствованную схему оценивания качества алгоритмов, а именно все данные нужно будет делить на 3 части (в случае использования отложенной выборки): обучение, валидация и контроль. Каждый из тысячи алгоритмов будет обучен на обучающей выборке, а его качество будет измерено на валидационной. Алгоритм с наилучшим качеством будет проверен на тестовой выборке, чтобы исключить переобучение и проверить алгоритм на адекватность. По сути именно тестовая выборка будет играть роль новых данных.

Если предпочтительно использовать кросс-валидацию, то данные следует разбить на 2 части. Первая из них будет использоваться для обучения алгоритмов и оценки качества с помощью кросс-валидации, после чего лучший алгоритм будет проверен на адекватность на контрольной выборке.

Урок 4

Метрики качества

4.1. Метрики качества в задачах регрессии

4.1.1. Применение метрик качества в машинном обучении

Метрики качества могут использоваться:

- Для задания функционала ошибки (используется при обучении).
- Для подбора гиперпараметров (используется при измерении качества на кросс-валидации). В том числе можно использовать другую метрику, которая отличается от метрики, с помощью которой построен функционал ошибки.
- Для оценивания итоговой модели: пригодна ли модель для решения задачи.

Далее мы рассмотрим, какие метрики можно использовать в задачах регрессии.

4.1.2. Среднеквадратичная ошибка

Первая метрика, о которой уже шла речь — среднеквадратичная ошибка:

$$MSE(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2.$$

Такой функционал легко оптимизировать, используя, например, метод градиентного спуска.

Этот функционал сильно штрафует за большие ошибки, так как отклонения возводятся в квадрат. Это приводит к тому, что штраф на выбросе будет очень сильным, и алгоритм будет настраиваться на выбросы. Другими словами, алгоритм будет настраиваться на такие объекты, на которые не имеет смысла настраиваться.

4.1.3. Средняя абсолютная ошибка

Похожий на предыдущий функционал качества — средняя абсолютная ошибка:

$$MAE(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} |a(x_i) - y_i|.$$

Этот функционал сложнее минимизировать, так как у модуля производная не существует в нуле. Но у такого функционала больше устойчивость к выбросам, так как штраф за сильное отклонение гораздо меньше.

4.1.4. Коэффициент детерминации

Коэффициент детерминации $R^2(a, X)$:

$$R^2(a, X) = 1 - \frac{\sum_{i=1}^{\ell} (a(x_i) - y_i)^2}{\sum_{i=1}^{\ell} (y_i - \bar{y})^2}, \quad \bar{y} = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i,$$

позволяет интерпретировать значение среднеквадратичной ошибки. Этот коэффициент показывает, какую долю дисперсии (разнообразия ответов) во всем целевом векторе y модель смогла объяснить.

Для разумных моделей коэффициент детерминации лежит в следующих пределах:

$$0 \leq R^2 \leq 1,$$

причем случай $R^2 = 1$ соответствует случаю идеальной модели, $R^2 = 0$ — модели на уровне оптимальной «константной», а $R^2 < 0$ — модели хуже «константной» (такие алгоритмы никогда не нужно рассматривать). Оптимальным константным алгоритмом называется такой алгоритм, который возвращает всегда среднее значение ответов \bar{y} для объектов обучающей выборки.

4.1.5. Несимметричные потери

До этого рассматривались симметричные модели, то есть такие, которые штрафуют как за недопрогноз, так и за перепрогноз. Но существуют такие задачи, в которых эти ошибки имеют разную цену.

Пусть, например, требуется оценить спрос на ноутбуки. В этом случае заниженный прогноз приведет к потере лояльности покупателей и потенциальной прибыли (будет закуплено недостаточное количество ноутбуков), а завышенный — только к не очень большим дополнительным расходам на хранение непроданных ноутбуков. Чтобы учесть это, функция потерь должна быть несимметричной и сильнее штрафовать за недопрогноз, чем за перепрогноз.

4.1.6. Квантильная ошибка

В таких случаях хорошо подходит квантильная ошибка или квантильная функция потерь:

$$\rho_\tau(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} \left((\tau - 1)[y_i < a(x_i)] + \tau[y_i \geq a(x_i)] \right) (y_i - a(x_i)).$$

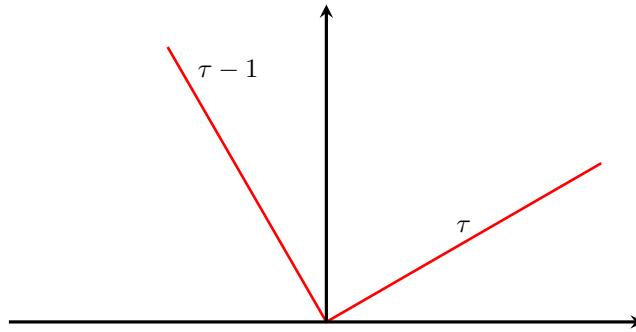


Рис. 4.1: График квантильной функции потерь

Параметр $\tau \in [0, 1]$ определяет то, за что нужно штрафовать сильнее — за недопрогноз или перепрогноз. Если τ ближе к 1, штраф будет больше за недопрогноз, а если, наоборот, ближе к 0 — за перепрогноз.

4.1.7. Вероятностный смысл квантильной ошибки

Чтобы разобраться, почему такая функция потерь называется квантильной, нужно разобраться с ее вероятностным смыслом. Пусть один и тот же объект x с одним и тем же признаковым описанием повторяется в выборке n раз, но на каждом из повторов — свой ответ y_1, \dots, y_n .

Такое может возникнуть при измерении роста человека. Измерения роста одного и того же человека могут отличаться ввиду ошибки прибора, а также зависеть от самого человека (может сгорбиться или выпрямиться).

При этом алгоритм должен для одного и того же признакового описания возвращать одинаковый прогноз. Другими словами, необходимо решить, какой прогноз оптимальен для x с точки зрения различных функционалов ошибки.

Оказывается, что если используется квадратичный функционал ошибки, то наиболее оптимальным прогнозом будет средний ответ на объектах, если абсолютный, то медиана ответов. Если же будет использоваться квантильная функция потерь, наиболее оптимальным прогнозом, будет τ -квантиль. В этом и состоит вероятностный смысл квантильной ошибки.

4.2. Метрика качества классификации

В этом блоке речь пойдет о том, как измерять качество в задачах классификации.

4.2.1. Доля правильных ответов

Как меру качества в задачах классификации естественно использовать долю неправильных ответов:

$$\frac{1}{\ell} \sum_{i=1}^{\ell} [a(x_i) \neq y_i]$$

Однако в задачах классификации принято выбирать метрики таким образом, чтобы их нужно было максимизировать, тогда как в задачах регрессии — так, чтобы их нужно было минимизировать. Поэтому определяют:

$$\text{accuracy}(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} [a(x_i) = y_i].$$

Эта метрика качества проста и широко используется, однако имеет несколько существенных недостатков.

4.2.2. Несбалансированные выборки

Первая проблема связана с несбалансированными выборками. Показателен следующий пример. Пусть в выборке 1000 объектов, из которых 950 относятся к классу -1 и 50 — к классу $+1$. Рассматривается бесполезный (поскольку не восстанавливает никаких закономерностей в данных) константный классификатор, который на всех объектах возвращает ответ -1 . Но доля правильных ответов на этих данных будет равна 0.95, что несколько много для бесполезного классификатора.

Чтобы «бороться» с этой проблемой, используется следующий факт. Пусть q_0 — доля объектов самого крупного класса, тогда доля правильных ответов для разумных алгоритмов $\text{accuracy} \in [q_0, 1]$, а не $[1/2, 1]$, как это можно было бы ожидать. Поэтому, если получается высокий процент правильных ответов, это может быть связано не с тем, что построен хороший классификатор, а с тем, что какого-то класса сильно больше, чем остальных.

4.2.3. Цены ошибок

Вторая проблема с долей верных ответов состоит в том, что она никак не учитывает разные цены разных типов ошибок. Тогда как цены действительно могут быть разными.

Например, в задаче кредитного scoringа, то есть в задаче принятия решения относительно выдачи кредита, сравниваются две модели. При использовании первой модели кредит будет выдан 100 клиентам, 80 из которых его вернут. Во второй модели, более консервативной, кредит был выдан только 50 клиентам, причем вернули его в 48 случаях. То, какая из двух моделей лучше, зависит от того, цена какой из ошибок выше: не дать кредит клиенту, который мог бы его вернуть, или выдать кредит клиенту, который его не вернет. Таким образом, нужны дополнительные метрики качества, которые учитывают цены той или иной ошибки.

4.3. Точность и полнота

4.3.1. Цены ошибок

В этом разделе пойдет речь о метриках качества классификации, которые позволяют учитывать разные цены ошибок. В конце предыдущего раздела уже было сказано, что цены ошибок действительно могут быть разными. Так, в задаче банковского scoringа необходимо принять решение, что хуже: выдать кредит «плохому» клиенту или не выдать кредит «хорошему» клиенту. Доля верных ответов не способна учитывать цены разных ошибок и поэтому не может дать ответа на этот вопрос.

4.3.2. Матрица ошибок

Удобно классифицировать различные случаи, как соотносятся между собой результат работы алгоритма и истинный ответ, с помощью так называемой матрицы ошибок.

	$y = 1$	$y = -1$
$a(x) = 1$	True Positive (TP)	False Positive (FP)
$a(x) = -1$	False Negative (FN)	True Negative (TN)

Когда алгоритм относит объект к классу $+1$, говорят, что алгоритм срабатывает. Если алгоритм сработал и объект действительно относится к классу $+1$, имеет место верное срабатывание (true positive), а если объект на самом деле относится к классу -1 , имеет место ложное срабатывание (false positive).

Если алгоритм дает ответ -1 , говорят, что он пропускает объект. Если имеет место пропуск объекта класса $+1$, то это ложный пропуск (false negative). Если же алгоритм пропускает объект класса -1 , имеет место истинный пропуск (true negative).

Таким образом, существуют два вида ошибок: ложные срабатывания и ложные пропуски. Для каждого из них нужна своя метрика качества, чтобы измерить, какое количество ошибок какого типа совершаются.

4.3.3. Точность и полнота

Пусть для примера рассматриваются две модели $a_1(x)$ и $a_2(x)$. Выборка состоит из 200 объектов, из которых 100 относятся к классу 1 и 100 — к классу -1 . Матрицы ошибок имеют вид:

	$y = 1$	$y = -1$		$y = 1$	$y = -1$
$a_1(x) = 1$	80	20	$a_2(x) = 1$	48	2
$a_1(x) = -1$	20	80	$a_2(x) = -1$	52	98

Введем две метрики. Первая метрика, точность (precision), показывает, насколько можно доверять классификатору в случае срабатывания:

$$precision(a, X) = \frac{TP}{TP + FP}.$$

Вторая метрика, полнота (recall), показывает, на какой доле истинных объектов первого класса алгоритм срабатывает:

$$recall(a, X) = \frac{TP}{TP + FN}.$$

В примере выше точность и полнота первого алгоритма оказываются равными:

$$\begin{aligned} precision(a_1, X) &= 0.8, & precision(a_2, X) &= 0.96 \\ recall(a_1, X) &= 0.8, & recall(a_2, X) &= 0.48 \end{aligned}$$

Вторая модель является очень точной, но в ущерб полноте.

4.3.4. Примеры использования точности и полноты

Первый пример — использование в задаче кредитного scoringa. Пусть в задаче кредитного scoringa ставится условие, что неудачных кредитов должно быть не больше 5%. В таком случае задача является задачей максимизации полноты при условии $precision(a, X) \geq 0.95$.

Второй пример — использование в медицинской диагностике. Необходимо построить модель, которая определяет, есть или нет определенное заболевание у пациента. При этом требуется, чтобы были выявлены как минимум 80% пациентов, которые действительно имеют данное заболевание. Тогда ставят задачу максимизации точности при условии $recall(a, X) \geq 0.8$.

Следует особо обратить внимание на то, как точность и полнота работают в случае несбалансированных выборок. Пусть рассматривается выборка со следующей матрицей ошибок:

	$y = 1$	$y = -1$
$a(x) = 1$	10	20
$a(x) = -1$	90	10000

Доля верных ответов (accuracy), точность(precision) и полнота(recall) для данного случая:

$$\text{accuracy}(a, X) = 0.99, \quad \text{precision}(a, X) = 0.33, \quad \text{recall}(a, X) = 0.1.$$

То, что доля верных ответов равняется 0.99, ни о чем не говорит: алгоритм все равно делает 66% ложных срабатываний и выявляет только 10% положительных случаев. Благодаря введению точности и полноты становится понятно, что алгоритм нужно улучшать.

4.4. Объединение точности и полноты

В этом разделе пойдет речь о том, как соединить точность и полноту в одну метрику качества классификации.

4.4.1. Точность и полнота (напоминание)

Точность показывает, насколько можно доверять классификатору в случае срабатывания:

$$\text{precision}(a, X) = \frac{TP}{TP + FP}.$$

Полнота показывает, на какой доле истинных объектов первого класса алгоритм срабатывает:

$$\text{recall}(a, X) = \frac{TP}{TP + FN}.$$

В некоторых задачах есть ограничения на одну из этих метрик, тогда как по второй метрике будет производиться оптимизация. Но в некоторых случаях хочется максимизировать и точность, и полноту одновременно. Встает вопрос об объединении этих двух метрик.

4.4.2. Арифметическое среднее

Единая метрика может быть получена как арифметическое среднее точности и полноты:

$$A = \frac{1}{2}(\text{precision} + \text{recall})$$

Пусть есть алгоритм, точность которого равна 10%, а полнота — 100%:

$$\text{precision} = 0.1 \quad \text{recall} = 1.$$

Это может быть случай, когда в выборке всего 10% объектов класса +1, а алгоритм является константным и всегда возвращает +1. Очевидно, что этот алгоритм плохой, но введенная выше метрика для него равна $A = 0.55$. В свою очередь другой, гораздо более лучший алгоритм, с $\text{precision} = 0.55$ и $\text{recall} = 55$ также характеризуется $A = 0.55$.

Ситуация, когда константный и разумный алгоритмы могут лежать на одной линии, является недопустимой, поэтому следует искать другой способ построения единой метрики.

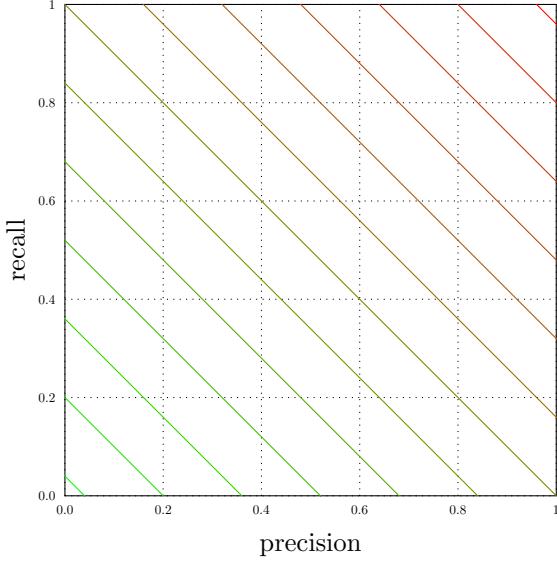


Рис. 4.2: Линии $A = \frac{1}{2}(precision + recall) = const$ в координатах precision–recall

4.4.3. Минимум

Чтобы конструктный и разумный алгоритмы не лежали на одной линии уровня, можно рассматривать:

$$M = \min(precision, recall).$$

Данный подход решает вышеупомянутую проблему, например:

$$precision = 0.05, \quad recall = 1 \quad \Rightarrow \quad M = 0.05.$$

Но есть другой нюанс: два алгоритма, для которых точности одинаковы, но отличаются значениями полноты, будут лежать на одной линии уровня M :

$$\begin{aligned} precision = 0.4, \quad recall = 0.5 &\quad \Rightarrow \quad M = 0.4, \\ precision = 0.4, \quad recall = 0.9 &\quad \Rightarrow \quad M = 0.4. \end{aligned}$$

Такое тоже недопустимо, так как второй алгоритм существенно лучше первого.

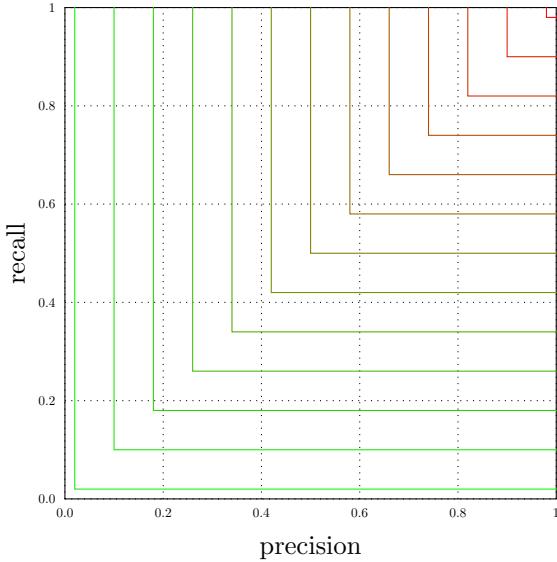


Рис. 4.3: Линии $M = \min(precision, recall) = const$ в координатах precision–recall

4.4.4. F-мера

«Сгладить» минимум можно с помощью гармонического среднего, или F -меры:

$$F = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

Для двух упомянутых выше алгоритма значения F -меры, в отличие от M , будут отличаться:

$$\begin{aligned} \text{precision} &= 0.4, & \text{recall} &= 0.5 \quad \Rightarrow \quad F = 0.44, \\ \text{precision} &= 0.4, & \text{recall} &= 0.9 \quad \Rightarrow \quad F = 0.55. \end{aligned}$$

Если необходимо отдать предпочтение точности или полноте, следует использовать расширенную F -меру, в которой есть параметр β :

$$F = (1 + \beta^2) \frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}.$$

Например, при $\beta = 0.5$ важнее оказывается полнота, а в случае $\beta = 2$, наоборот, важнее оказывается точность.

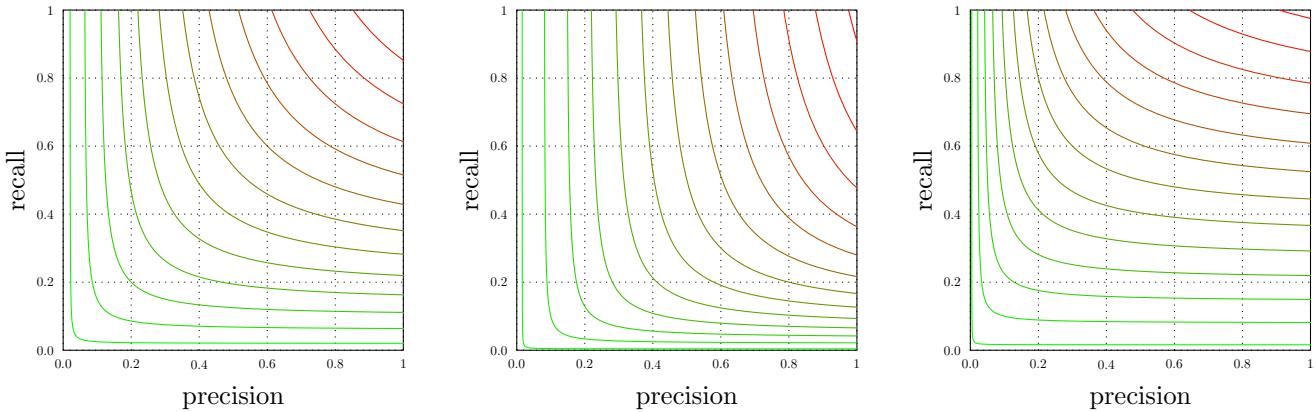


Рис. 4.4: Линии $F = \text{const}$ в координатах precision–recall при значениях $\beta = 1$, $\beta = 0.5$ и $\beta = 2$ соответственно

4.5. Качество оценок принадлежности классу

4.5.1. Оценка принадлежности

Многие алгоритмы бинарной классификации устроены следующим образом: сначала вычисляется некоторое вещественное число $b(x)$, которое сравнивается с порогом t .

$$a(x) = [b(x) > t],$$

где $b(x)$ — оценка принадлежности классу +1. Другими словами, $b(x)$ выступает в роли некоторой оценки уверенности, что x принадлежит классу +1.

В случае линейного классификатора $a(x) = [\langle w, x \rangle > t]$ оценка принадлежности классу +1 имеет вид $b(x) = \langle w, x \rangle$.

Часто бывает необходимо оценить качество именно оценки принадлежности, а порог выбирается позже из соображений на точность или полноту.

4.5.2. Оценка принадлежности в задаче кредитного скоринга

Пусть рассматривается задачного кредитного скоринга и была построена некоторая функция $b(x)$, которая оценивает вероятность возврата кредита клиентом x . Далее классификатор строится следующим образом:

$$a(x) = [b(x) > 0.5]$$

При этом получилось, что точность (precision) равна 10%, а полнота (recall) — 70%. Это очень плохой алгоритм, так как 90% клиентов, которым будет выдан кредит, не вернут его.

При этом не понятно, в чем дело: был плохо выбран порог или алгоритм не подходит для решения данной задачи. Именно для этого необходимо измерять качество самих оценок $b(x)$.

4.5.3. PR-кривая

Первый способ оценки принадлежности классу основан на использовании кривой точности-полноты. По оси X откладывается полнота, а по оси Y — точность. Каждой точке на этой кривой будет соответствовать классификатор с некоторым значением порога.

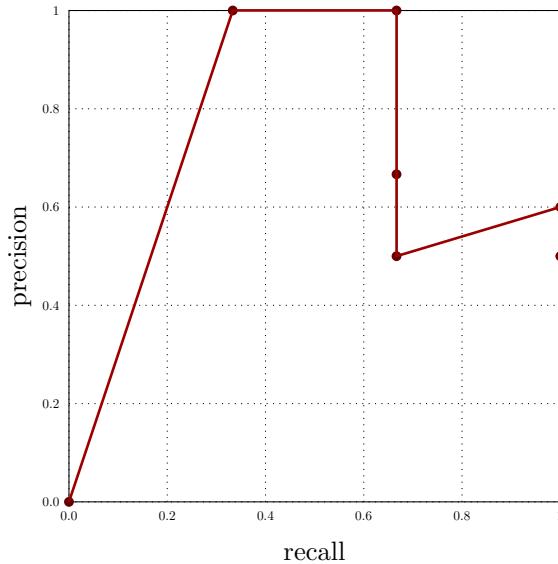


Рис. 4.5: Кривая полноты–точности

Для примера будет приведено построение PR-кривой для выборки из 6 объектов, три из которых относятся к классу 1 и 3 — к классу 0. Соответствующий ей график изображен выше.

b(x)	0.14	0.23	0.39	0.54	0.73	0.90
y	0	1	0	0	1	1

- При достаточно большом пороге ни один объект не будет отнесен к классу 1. В этом случае и точность и полнота равны 0.
- При таком пороге, что ровно один объект отнесен к классу 1, точность будет 100% (поскольку этот объект действительно из 1 класса), а полнота — $1/3$ (поскольку всего 3 объекта 1 класса).
- При дальнейшем уменьшении порога уже два объекта отнесены к классу 1, точность также остается 100%, а полнота становится равной $2/3$.
- При таком пороге, что уже три объекта будут отнесены к классу 1, точность становится равной $2/3$, а полнота остается такой же.
- При таком пороге, что четыре объекта отнесены к классу 1, точность уменьшится до 0.5, а полнота опять не изменится.
- При дальнейшем уменьшении порога уже 5 объектов будут отнесены к 1 классу, полнота станет равной 100%, а точность — $3/5$.

В реальных задачах с числом объектов порядка нескольких тысяч или десятков тысяч, кривая точности-полноты выглядит примерно следующим образом.

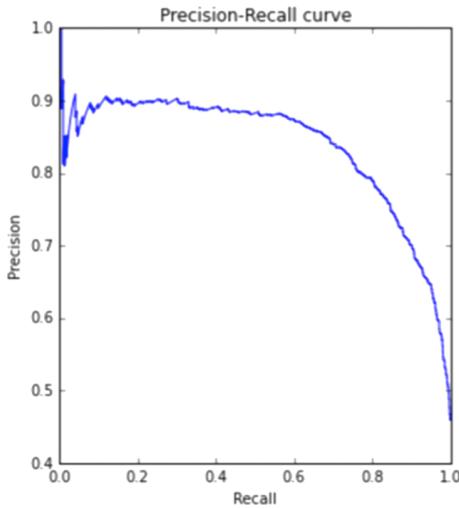


Рис. 4.6: Кривая точности-полноты в реальных задачах с десятками тысяч объектов

Следует отметить, что начинается PR-кривая всегда из точки $(0, 0)$, а заканчивается точной $(1, r)$, где r — доля объектов класса 1.

В случае идеального классификатора, то есть если существует такой порог, что и точность, и полнота равны 100%, кривая будет проходить через точку $(1, 1)$. Таким образом, чем ближе кривая пройдет к этой точке, тем лучше оценки. Площадь под этой кривой может быть хорошей мерой качества оценок принадлежности к классу 1. Такая метрика называется AUC-PRC, или площадь под PR-кривой.

4.5.4. ROC-кривая

Второй способ измерить качество оценок принадлежности к классу 1 — ROC-кривая, которая строится в осях False Positive Rate (ось X) и True Positive Rate (ось Y):

$$FPR = \frac{FP}{FP + TN}, \quad TPR = \frac{TP}{TP + FN}.$$

ROC-кривая строится аналогично PR-кривой: постепенно рассматриваются случаи различных значений порогов и отмечаются точки на графике. Для упомянутой выше выборки ROC-кривая имеет следующий вид:

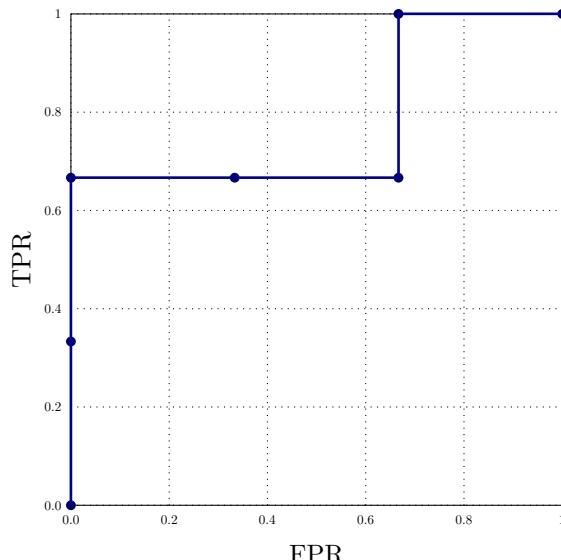


Рис. 4.7: ROC-кривая

В случае с большой выборкой ROC-кривая выглядит следующим образом:

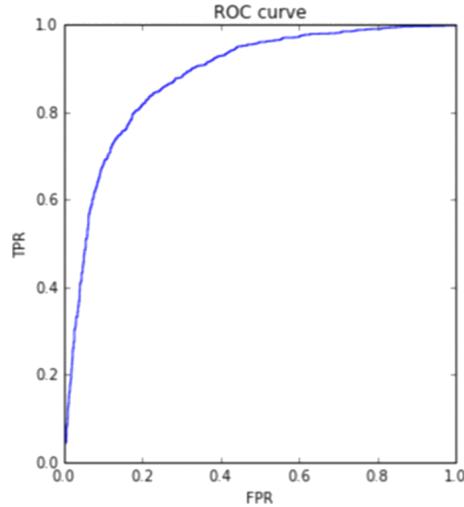


Рис. 4.8: Кривая ROC в реальных задачах с десятками тысяч объектов

Кривая стартует с точки $(0, 0)$ и приходит в точку $(1, 1)$. При этом, если существует идеальный классификатор, кривая должна пройти через точку $(0, 1)$. Чем ближе кривая к этой точке, тем лучше будут оценки, а площадь под кривой будет характеризовать качество оценок принадлежности к первому классу. Такая метрика называется AUC-ROC, или площадь под ROC-кривой.

4.5.5. Особенности AUC-ROC

Как было написано выше, ROC-кривая строится в осях FPR и TPR , которые нормируются на размеры классов:

$$FPR = \frac{FP}{FP + TN}, \quad TPR = \frac{TP}{TP + FN}.$$

Следовательно, при изменении баланса классов величина AUC-ROC и неизменных свойствах объектов выборки площадь под ROC-кривой не изменится. В случае идеального алгоритма $AUC - ROC = 1$, а в случае худшего $AUC - ROC = \frac{1}{2}$.

Значение $AUC - ROC$ имеет смысл вероятности того, что если были выбраны случайный положительный и случайный отрицательный объекты выборки, положительный объект получит оценку принадлежности выше, чем отрицательный объект.

4.5.6. Особенности AUC-PRC

PR-кривая строится в осях precision и recall:

$$precision = \frac{TP}{TP + FP}, \quad recall = \frac{TP}{TP + FN},$$

а следовательно изменяется при изменении баланса классов.

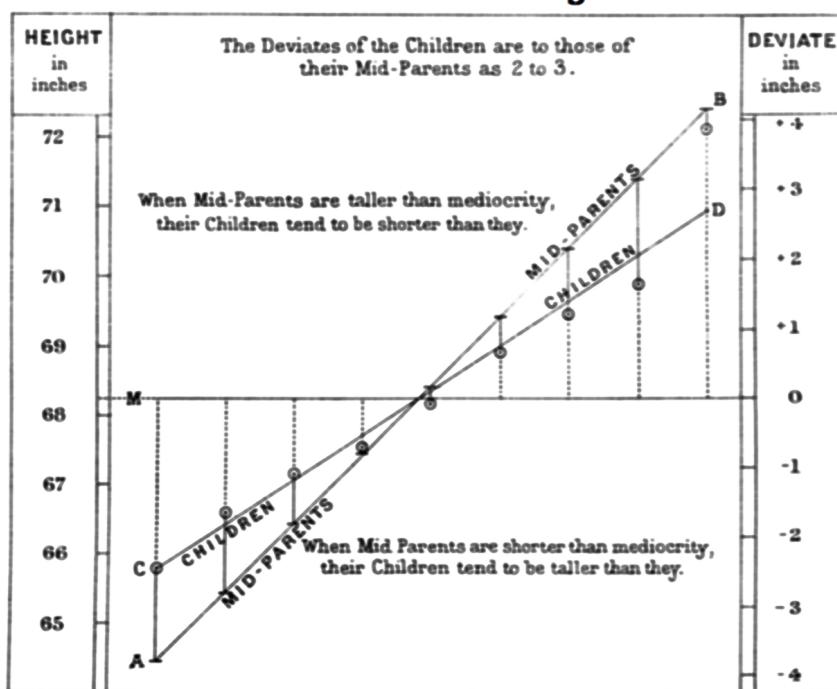
Урок 5

Линейные модели: статистический взгляд

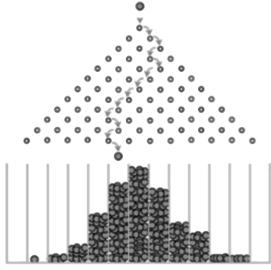
5.1. Задача регрессии

5.1.1. История термина

Впервые термин регрессия появился в конце XIX века в работе Френсиса Гальтона:



В этой работе «Регрессия к середине в наследственности роста» Френсис Гальтон исследовал зависимость между средним ростом детей и средним ростом их родителей и обнаружил, что отклонение роста детей от среднего составляет примерно $\frac{2}{3}$ отклонения роста родителей от среднего. Казалось бы, со временем люди должны рождаться все ближе и ближе к среднему росту. На самом деле, естественно, этого не происходит.



Лучше понять эффект регрессии к среднему позволяет другое творение Френсиса Гальтона, которое называется машина, или доска, Гальтона.

Это механическая машина, в которой сверху в центральной части находятся шарики. Когда открывается заслонка, шарики начинают постепенносыпаться вниз, ударяясь о штырьки, которые расположены на одинаковом расстоянии друг от друга. При каждом соударении шарика со штырьком вероятности того, что он упадет налево и направо от штырька, равны. Постепенно шарики начинают собираться в секциях внизу в гауссиану, или плотность нормального распределения.

Чтобы понять эффект регрессии к среднему, давайте мысленно подставим к машине Гальтона снизу еще одну такую же машину. Если теперь убрать перегородку, которая удерживает шарики в верхней половине, они начнут постепенно осыпаться вниз и сформируют внизу еще одну такую же гауссиану. Если зафиксировать какой-то конкретный шарик в нижней половине ближе к краю, то откажется, что с достаточно большой вероятностью этот шарик пришел не из ячейки, которая находится в верхней половине прямо над ячейкой, в которой он оказался внизу, а от ячейки ближе к середине. Это происходит просто потому, что в середине шариков больше.

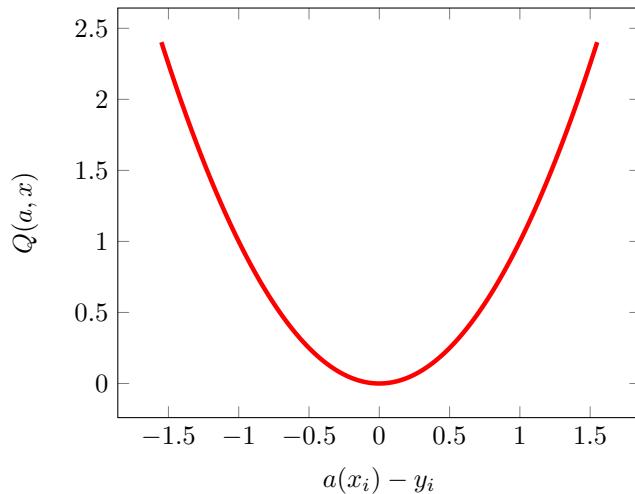
Эффект регрессии к среднему проявляется во многих практических задачах. Например, если дать студентам очень сложный тест, большую роль в том, насколько хорошо они его пройдут, будут играть не только их знания по предмету, но и везение, то есть случайный фактор. Поэтому, если изолировать 10% студентов, которые прошли тест лучше всех (набрали больше всего баллов) и дать им еще один вариант теста, то средний балл в этой группе скорее всего упадет. Просто потому что люди, которым повезло в первый раз, скорее всего уже не будут так удачливы во второй — в этом и состоит эффект регрессии к середине.

Френсис Гальтон был основоположником дактилоскопии, исследовал явление синестезии, внес существенный вклад в метеорологию, впервые описав циклоны и антициклоны, а также, например, изобрел ультразвуковой свисток для собак. Но именно регрессия и по сей день остается одним из наиболее важнейших инструментов, к которому он приложил руку.

5.1.2. Регрессия

Чаще всего под регрессией понимают минимизацию среднеквадратичной ошибки: квадратов отклонений откликов у от их предсказанных значений $a(x)$.

$$Q(w, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (\langle w, x_i \rangle - y_i)^2$$



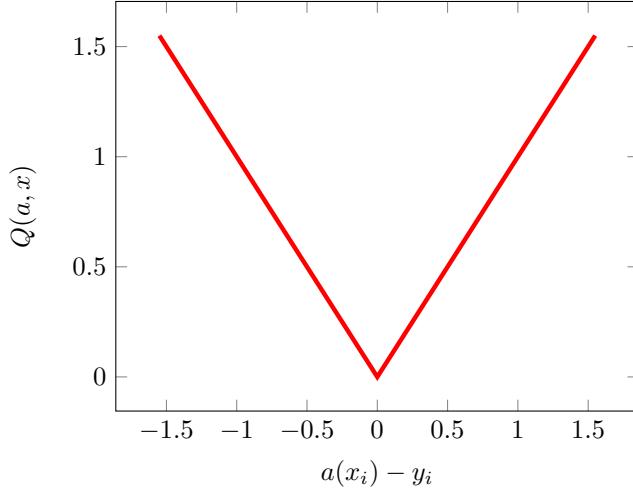
Поскольку минимизируется сумма квадратов отклонений, этот метод называется методом наименьших квадратов (сокращенно МНК). Для линейной регрессии задача имеет аналитическое решение.

$$w_*(x) = \operatorname{argmin}_w Q(w, X) = (X^T X)^{-1} X^T y.$$

Именно этим частично объясняется популярность среднеквадратичной ошибки.

В *XIX* веке, когда эта задача впервые возникла, никакого способа ее решения, кроме аналитического, быть не могло. Сейчас можно численно минимизировать не только среднеквадратичную ошибку, но и, например, среднюю абсолютную, то есть сумму модулей отклонений нашей модели от отклика:

$$Q(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} |a(x_i) - y_i|, \quad a_*(x) = \operatorname{argmin}_a Q(a, X).$$



Такая задача является частным случаем класса задач квантильной регрессии.

5.2. Метод максимизации правдоподобия

Пусть x — некоторая случайная величина с функцией распределения $F(x, \theta)$, которая зависит от неизвестного параметра θ , а $X^n = (X_1, \dots, X_n)$ — выборка размера n , сгенерированная из распределения $F(x, \theta)$. Необходимо оценить по данной выборке неизвестный параметр.

5.2.1. Метод максимизации правдоподобия: пример

Чтобы понять метод максимального правдоподобия, можно рассмотреть еще один исторический пример. Эти данные собраны в конце *XIX* века. В Генеральный штаб прусской армии ежегодно в течение 20 лет от десяти кавалерийских корпусов поступали данные о количестве смертей кавалеристов в результате гибели под ними коня.

Кол-во погибших	0	1	2	3	4	5	Всего
Кол-во донесений	109	65	22	3	1	0	200

Поскольку данная случайная величина — счетчик, ее необходимо моделировать распределением Пуассона, функция вероятности которого имеет вид:

$$P(X = k) = \frac{\lambda^k e^{-\lambda}}{k!}$$

Поскольку выборка состоит из независимых, одинаково распределенных случайных величин, вероятность получения строго определенной выборки равна произведению вероятностей получения каждого из элементов этой выборки:

$$P(X^n, \lambda) = \prod_{i=1}^n \frac{\lambda^{X_i} e^{-\lambda}}{X_i!} \equiv L(X^n, \lambda).$$

Функция L зависит от неизвестного параметра λ и называется правдоподобием выборки. В качестве оценки для λ можно взять такое значение, которое максимизирует функцию правдоподобия:

$$\hat{\lambda}_{\text{ОМП}} = \operatorname{argmax}_\lambda L(X^N, \lambda)$$

Эта оценка называется оценкой максимального правдоподобия.

Несложно показать, что в рассматриваемой задаче оценка максимального правдоподобия для параметра λ совпадает с выборочным средним:

$$\hat{\lambda}_{\text{ОМП}} = \bar{X}_n = 0.61$$

Чтобы это сделать, можно взять производную от логарифма функции правдоподобия и приравнять ее к нулю. Здесь используется тот факт, что логарифмирование L не меняет точку максимума функции.

5.2.2. Метод максимизации правдоподобия: общий вид

В общем виде метод максимума правдоподобия записывается следующим образом. Пусть некоторая случайная величина x имеет распределение $F(x, \theta)$, X^n — выборка размера n :

$$X \sim F(x, \theta), \quad X^n = (X_1, \dots, X_n),$$

Тогда функция правдоподобия имеет вид:

$$L(X^n, \lambda) = \prod_{i=1}^n P(X = X_i, \theta).$$

Поскольку при логарифмировании не меняются положения максимумов функции, удобно работать не с самим правдоподобием, а с логарифмом правдоподобия:

$$\ln L(X^n, \lambda) = \sum_{i=1}^n \ln P(X = X_i, \theta).$$

Оценкой максимального правдоподобия называется величина:

$$\hat{\lambda}_{\text{ОМП}} = \operatorname{argmax}_\lambda \ln L(X^N, \lambda)$$

В случае непрерывной случайной величины метод максимального правдоподобия записывается аналогично:

$$X \sim F(x, \theta), \quad L(X^n, \lambda) = \prod_{i=1}^n f(X = X_i, \theta), \quad \hat{\lambda}_{\text{ОМП}} = \operatorname{argmax}_\lambda L(X^N, \lambda).$$

5.2.3. Свойства метода максимального правдоподобия

Метод максимального правдоподобия обладает рядом очень полезных свойств:

- Состоятельность, то есть получаемые оценки при увеличении объема выборки начинают стремиться к истинным значениям:

$$\hat{\lambda}_{\text{ОМП}} \rightarrow \theta \quad \text{при} \quad n \rightarrow \infty.$$

- Асимптотическая нормальность, то есть с ростом объема выборки, оценки максимального правдоподобия все лучше описываются нормальным распределением с средним, равным истинному значению θ , и дисперсией, равной величине, обратной к информации Фишера:

$$\hat{\lambda}_{\text{ОМП}} \sim N(\theta, I^{-1}(\theta)) \quad \text{при} \quad n \rightarrow \infty.$$

5.3. Регрессия как максимизация правдоподобия

5.3.1. Модель шума: нормальное распределение

При решении задачи регрессии значение отклика приближается в виде

$$y = a(x) + \varepsilon,$$

где $a(x)$ — регрессионная функция, а компонента ε описывает случайный шум.

Если этот случайный шум имеет нормальное распределение с нулевым средним и дисперсией σ^2 , оказывается, что задача минимизации среднеквадратичной ошибки

$$a_* = \operatorname{argmin}_a \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2$$

дает оценку максимального правдоподобия для регрессионной функции $a(x)$.

Этот факт позволяет использовать в задаче с регрессии свойства метода максимального правдоподобия. Например, используя асимптотическую нормальность, можно определять значимость признаков x^j в модели и делать их отбор. Также можно строить доверительные интервалы для значения отклика на новых объектах, которых нет в обучающей выборке.

5.3.2. Модель шума: распределение Лапласа

Распределение шума не обязательно должно быть нормальным и может быть каким-то другим. Например, можно попытаться описать его распределением Лапласа с нулевым средним. Формула для функции плотности вероятности такого распределения:

$$f(x) = \frac{\alpha}{2} e^{-\alpha|x|}.$$

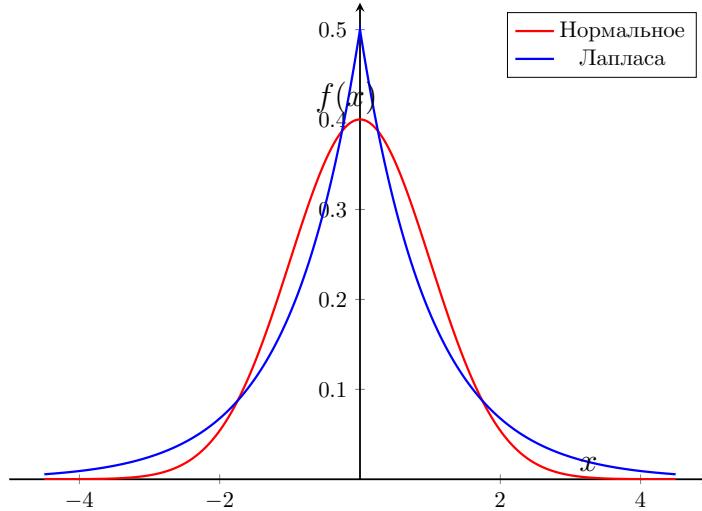


Рис. 5.1: График функции Лапласа с нулевым средним и нормального распределения.

По сравнению с нормальным распределением, распределение Лапласа имеет более тяжелые хвосты, то есть для него более вероятны большие значения ε . Другими словами, если моделировать шум распределением Лапласа, то наблюдения могут сильнее отклоняться от выбранной модели. За счет этого получается решение, которое более устойчивое к выбросам. Оказывается, что если шум действительно описывается распределением Лапласа, то к оценке максимального правдоподобия приводит минимизация средних абсолютных отклонений:

$$a_* = \operatorname{argmin}_a \frac{1}{\ell} \sum_{i=1}^{\ell} |a(x_i) - y_i|.$$

5.4. Регрессия как оценка среднего

5.4.1. Среднеквадратичная ошибка

Пусть для начала a — константа (что соответствует ситуации, когда отсутствуют признаки), а y является случайной функцией с плотностью распределения $f(t)$. В таком случае среднеквадратичная ошибка имеет вид:

$$Q(a) = \int_t (a - t)^2 f(t) dt.$$

Нетрудно показать, что:

$$a_* = \operatorname{argmin}_a Q(a) = \mathbb{E}y,$$

то есть наилучшая константа, которая аппроксимирует значение y в смысле среднеквадратичной ошибки — это математическое ожидание.

Если $a(x)$ — произвольная функция признаков x , функционал среднеквадратичной ошибки имеет вид:

$$Q(a, X) = \int_t (a(x) - t)^2 f(t) dt,$$

а его минимум будет доставлять условное математическое ожидание:

$$a_*(x) = \operatorname{argmin}_a Q(a(x)) = \mathbb{E}(y|x).$$

Таким образом, в случае с конечной выборкой:

$$Q(a(x), X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2,$$

оценка, получаемая при минимизации среднеквадратичной ошибки:

$$a_*(x) = \operatorname{argmin}_a Q(a, X)$$

является лучшей аппроксимацией условного математического ожидания $\mathbb{E}(y|x)$. В случае линейной регрессии, то есть когда отклик моделируется линейной комбинацией $\langle w, x_i \rangle$:

$$Q(w, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (\langle w, x_i \rangle - y_i)^2, \quad w_* = \operatorname{argmin}_w Q(w, X),$$

выражение $\langle w_*, x_i \rangle$ является наилучшей линейной аппроксимацией условного математического ожидания $\mathbb{E}(y|x)$.

Полученный результат согласуется с интуитивными представлениями. Действительно, пусть $y_i = 2$, график зависимости ошибки на этом объекте в зависимости от предсказания алгоритма $a(x)$ выглядит следующим образом.

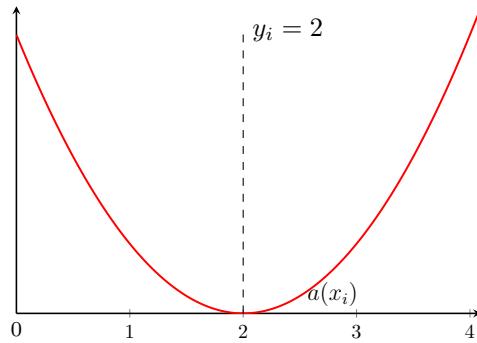


Рис. 5.2: Зависимость ошибки от предсказания алгоритма в случае среднеквадратичной ошибки.

По графику видно, что одинаково штрафуются отклонения предсказания как в большую, так и в меньшую сторону от истинного значения y_i . Поэтому не удивительно, что функция, которая доставляет минимум функции, представляет собой какое-то среднее.

5.4.2. Дивергенции Брегмана

Однако оказывается, что условное математическое ожидание доставляет минимум не только среднеквадратичной ошибки, но и более широкого класса функций потерь, которые называются дивергенциями Брегмана.

Дивергенции Брегмана порождаются любой непрерывной дифференцируемой выпуклой функцией φ :

$$Q(a, X) = \varphi(y) - \varphi(a(X)) - \varphi'(a(X))(y - a(X)).$$

Среднеквадратичная ошибка является частным случаем дивергенции Брегмана. Минимизируя любую дивергенцию Брегмана, мы получаем оценку для условного математического ожидания:

$$a_* = \operatorname{argmin}_a Q(a, X) — \text{лучшая аппроксимация } \mathbb{E}(y|x).$$

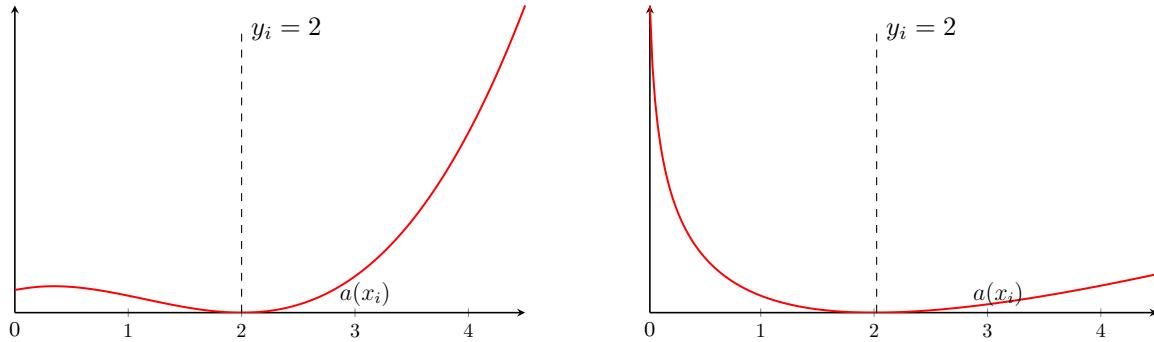


Рис. 5.3: Несколько функции потерь из класса дивергенций Брегмана.

Этот результат уже является несколько странным, поскольку в семействе дивергенций Брегмана можно найти, в том числе, несимметричные относительно y функции. Такие функции больше штрафуют за отклонение модели в большую или меньшую сторону. Это результат может быть несколько континтуитивным и получен не так давно.

5.4.3. Средняя абсолютная ошибка и несимметричная абсолютная ошибка

Средняя абсолютная ошибка:

$$Q(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} |a(x_i) - y_i|$$

не входит в семейство дивергенций Брегмана. При ее минимизации получается оценка не условного математического ожидания, а оценка условной медианы:

$$a_* = \operatorname{argmin}_a Q(a, X) — \text{лучшая аппроксимация } \operatorname{med}(y|x).$$

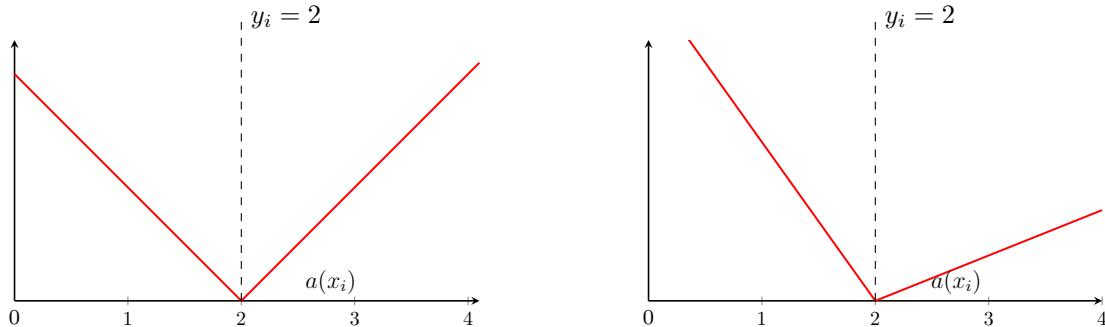


Рис. 5.4: Графики симметричной средней абсолютной и несимметричной абсолютной функций ошибок.

Несимметричная абсолютная функция ошибки («несимметричность» определяется параметром τ):

$$Q(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} ((\tau - 1)[y_i < a(x_i)] + \tau[y_i \geq a(x_i)]) (y_i - a(x_i))$$

имеет в некотором смысле наклоненный график по сравнению с графиком симметричной абсолютной ошибки. При минимизации такого функционала получается лучшая оценка для соответствующего условного квантиля:

$$a_* = \operatorname{argmin}_a Q(a, X) — \text{лучшая аппроксимация } y|x \text{ порядка } \tau.$$

5.5. Регуляризация

5.5.1. Переобучение регрессионных моделей

Если используется слишком сложная модель, а данных недостаточно, чтобы точно определить ее параметры, эта модель легко может получиться переобученной, то есть хорошо описывать обучающую выборку и плохо — тестовую. Бороться с этим можно различными способами:

- **Взять больше данных.** Такой вариант обычно недоступен, поскольку дополнительные данные стоят дополнительных денег, а также иногда недоступны совсем. Например, в задачах веб-поиска, несмотря на наличие терабайтов данных, эффективный объем выборки, описывающей персонализированные данные, существенно ограничен: в этом случае можно использовать только историю посещений данного пользователя.
- **Выбрать более простую модель** или упростить модель, например исключив из рассмотрения некоторые признаки. Процесс отбора признаков представляет собой нетривиальную задачу. В частности, не понятно, какой из двух похожих признаков следует оставлять, если признаки сильно зашумлены.
- **Использовать регуляризацию.** Ранее было показано, что у переобученной линейной модели значения весов в модели становятся огромными и разными по знаку. Если ограничить значения весов модели, то с переобучением можно до какой-то степени побороться.

5.5.2. L_1 -регуляризация и L_2 -регуляризация

Есть несколько способов провести регуляризацию:

- L_2 -регуляризатор (ridge-регрессия или гребневая регрессия):

$$w_* = \operatorname{argmin}_w \left(\frac{1}{\ell} \sum_{i=1}^{\ell} (\langle w, x_i \rangle - y_i)^2 + \lambda \sum_{j=1}^d w_j^2 \right).$$

- L_1 -регуляризатор (lasso-регрессия или лассо-регрессия):

$$w_* = \operatorname{argmin}_w \left(\frac{1}{\ell} \sum_{i=1}^{\ell} (\langle w, x_i \rangle - y_i)^2 + \lambda \sum_{j=1}^d |w_j| \right).$$

Понять различия между L_1 и L_2 регуляризаторами можно на модельном примере. Пусть матрица «объекты–признаки» X является единичной матрицей размера $\ell \times \ell$:

$$X = \begin{pmatrix} 1 & 0 & \dots & 0 \\ 0 & 1 & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & \dots & 1 \end{pmatrix}.$$

Тогда при решении задачи линейной регрессии использование метода наименьших квадратов без регуляризации:

$$w_* = \operatorname{argmin}_w \sum_{i=1}^{\ell} (w_i - y_i)^2,$$

дает следующий вектор весов:

$$w_{*j} = y_j$$

При использовании гребневой регуляризации (L_2 -регуляризация) компоненты вектора весов имеют вид:

$$w_{*j} = \frac{y_j}{1 + \lambda},$$

а при использовании L_1 -регуляризатора (lasso):

$$w_{*j} = \begin{cases} y_j - \lambda/2, & y_j > \lambda/2 \\ y_j + \lambda/2, & y_j < -\lambda/2 \\ 0, & |y_j| \leq \lambda/2. \end{cases}$$

При использовании только МНК без регуляризации $w_{*j} = y_j$. Соответствующая линия изображена пунктиром на обоих графиках. При использовании L_2 регуляризации зависимость w_{*j} от y_j все еще линейная, компоненты вектора весов ближе расположены к нулю.

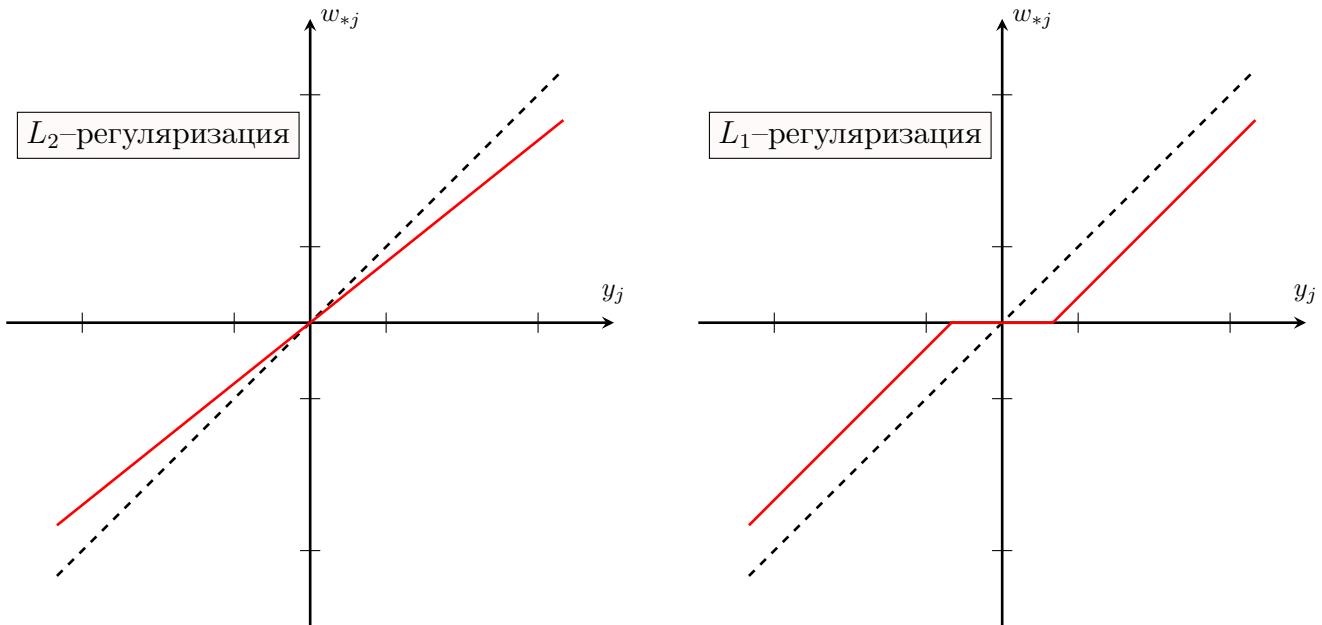


Рис. 5.5: Зависимость w_{*j} от значения отклика y_j при использовании различных регуляризаторов.

В случае L_1 регуляризации график выглядит несколько иначе: существует область (размера λ) значений y_j , для которых $w_j = 0$. То есть lasso, или L_1 -регуляризация, позволяет отбирать признаки, а именно: веса признаков, обладающих низкой предсказательной способностью, оказываются равными нулю.

5.5.3. Смещение и дисперсия

Можно показать, что математическое ожидание квадрата ошибки регрессии представляет собой сумму трех компонент:

$$\mathbb{E} (a_*(x) - y)^2 = \underbrace{(\mathbb{E} a_*(x) - a(x))^2}_{\text{Квадрат смещения}} + \underbrace{\text{Дисперсия оценки}}_{\text{Дисперсия оценки}} + \underbrace{\sigma^2}_{\text{Шум}}.$$

От выбора модели зависит квадрат смещения и дисперсия оценки, но не шум, который является свойством данных, а не модели.

Метод наименьших квадратов дает оценки, которые имеют нулевое смещение. Регуляризация позволяет получить смещенные оценки с меньшим $\mathbb{E} (a_*(x) - y)^2$ за счет того, что у этой оценки будет меньше дисперсия.

Следующая аналогия позволяет лучше понять баланс между смещением и дисперсией. При стрельбе по мишени среднее число набранных очков зависит от положения средней точки попадания и разбросом относительно этого среднего.

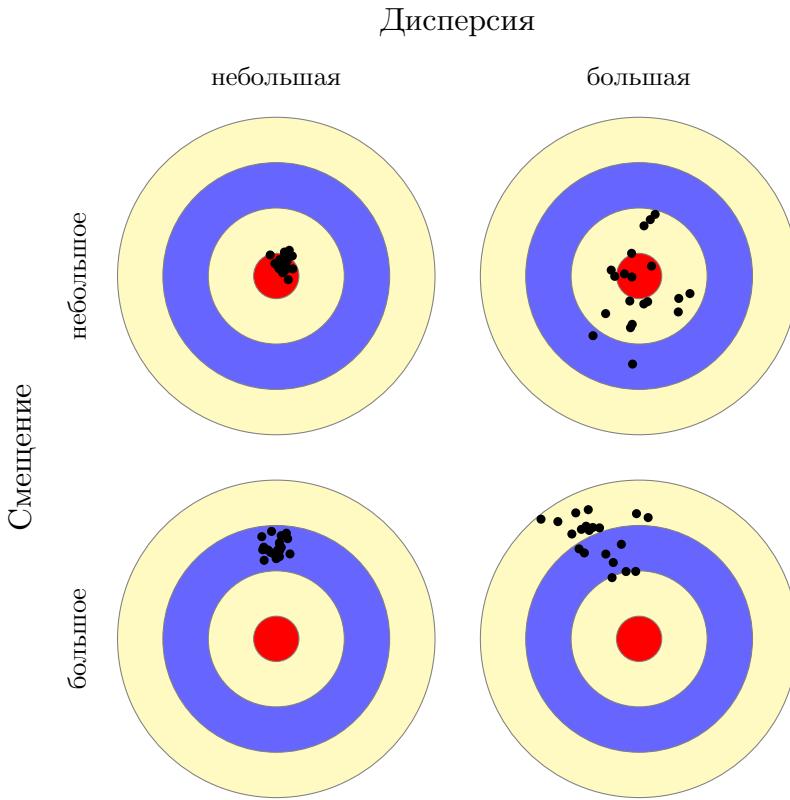


Рис. 5.6: Дисперсия и смещения при различных характеристиках стрельбы.

Лучший результат будет, если стрелять без смещения и без разброса. Переобучению линейных моделей соответствует стрельба без смещения, но с огромным разбросом. И часто оказывается, что можно набрать больше очков, стреляя не совсем в цель, то есть со смещением, но зато более точно. Именно это и позволяет добиться регуляризации.

5.5.4. Решение задач гребневой регрессии и лассо

В байесовской статистике гребневая регрессия соответствует заданию нормального априорного распределения на коэффициенты линейной модели, а метод лассо — заданию Лапласовского априорного распределения. Подробнее о байесовской статистике написано в соответствующем уроке.

Задача гребневой регрессии имеет аналитическое решение:

$$w_* = (X^T X + \lambda I)^{-1} X^T y.$$

Для решения задачи лассо аналитического решения не существует, однако есть очень эффективный численный способ получения решения.

5.6. Логистическая регрессия

5.6.1. Логистическая регрессия

Пусть \mathbb{X} — пространство объектов, \mathbb{Y} — пространство ответов, $X = (x_i, y_i)_{i=1}^\ell$ — обучающая выборка, $x = (x^1, \dots, x^d)$ — признаковое описание.

Логистическая регрессия — это метод обучения с учителем в задаче бинарной классификации $\mathbb{Y} = \{0, 1\}$.

5.6.2. Метод линейного дискриминанта Фишера

Метод линейного дискриминанта Фишера, один из самых старых методов классификации, заключается в минимизации среднеквадратичной ошибки:

$$Q(w, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (\langle w, x_i \rangle - y_i)^2.$$

В результате получается вектор весов:

$$w_* = \operatorname{argmin}_a Q(w, X),$$

причем, если для некоторого объекта $\langle w, x_i \rangle > 0.5$, объект относится к первому классу $y = 1$, в ином случае — к нулевому $y = 0$. На самом деле, хочется предсказывать не просто метки классов, а вероятности того, что объекты относятся к какому-то из классов:

$$P(y = 1|x) \equiv \pi(x).$$

Хотя $\pi(x)$ совпадает с условным математическим ожиданием $\mathbb{E}(y|x)$:

$$\pi(x) = 1 \cdot P(y = 1|x) + 0 \cdot P(y = 0|x) = \mathbb{E}(y|x),$$

использовать для оценки вероятности обычную линейную регрессию

$$\pi(x) \approx \langle w, x \rangle$$

не получится: получаемая линейная комбинация факторов не обязательно лежит на отрезке от 0 до 1.

Пусть, например, решается следующая задача. Необходимо предсказать вероятность невозврата платежа по кредитной карте в зависимости от размера задолженности.

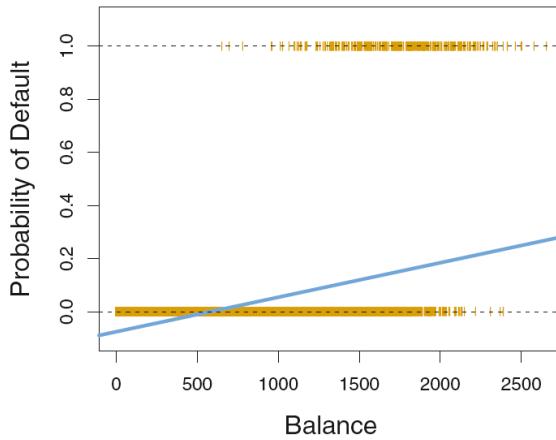


Рис. 5.7: Применение линейной регрессии в задаче оценки вероятности просрочки платежа.

По обучающей выборке была настроена модель линейной регрессии. Получается, что при задолженности 2000\$ вероятность просрочить платеж по кредиту равна 0.2, при задолженности 500\$ — нулю, а при меньших значениях и вовсе отрицательная. Также, если задолженность больше 10000\$, вероятность просрочки будет больше 1. Не понятно, как интерпретировать этот результат.

5.6.3. Обобщенные линейные модели

Пусть функция $g : (0, 1) \mapsto \mathbb{R}$ переводит интервал $(0, 1)$ на множество всех действительных чисел, тогда можно решать задачу линейной регрессии:

$$g(\mathbb{E}(y|x)) \approx \langle w, x \rangle,$$

в которой строится оценка не для условного матожидания $\mathbb{E}(y|x)$, а для $g(\mathbb{E}(y|x))$. Что то же самое:

$$\mathbb{E}(y|x) \approx g^{-1}(\langle w, x \rangle)$$

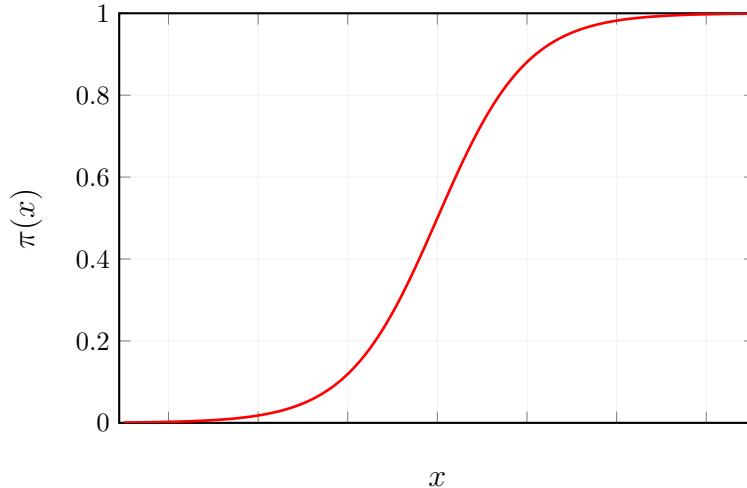
В статистике такое семейство моделей называется обобщенными линейными моделями.

В задаче бинарной классификации в качестве g^{-1} используется сигмоида:

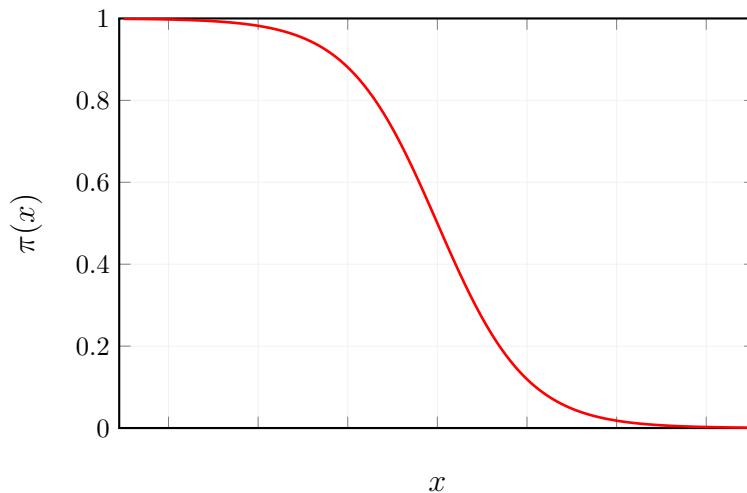
$$\pi(x) \approx \frac{e^{\langle w, x \rangle}}{1 + e^{\langle w, x \rangle}}.$$

В одномерном случае значение параметра w_0 сигмоиды определяет положение её центра на числовой оси, а w_1 — форму этой сигмоиды:

- Если $w_1 > 0$, сигмода возрастающая:



- Если $w_1 < 0$, сигмода убывающая:



Чем больше по модулю значение w_1 , тем круче наклон сигмоиды в области ее середины.

5.6.4. Предсказание вероятностей

Если использовать сигмоиду:

$$\pi(x) \approx \frac{e^{\langle w, x \rangle}}{1 + e^{\langle w, x \rangle}}$$

в обобщенной линейной модели в задаче логистической регрессии, результат будет более адекватным:

- Вероятность $\pi(x) \in [0, 1]$, как и требуется.

- На краях области значений x функция (вероятность) $\pi(x)$ слабо меняется при небольших изменениях x , когда как существенно изменяется, если x находится в середине диапазона своих значений.

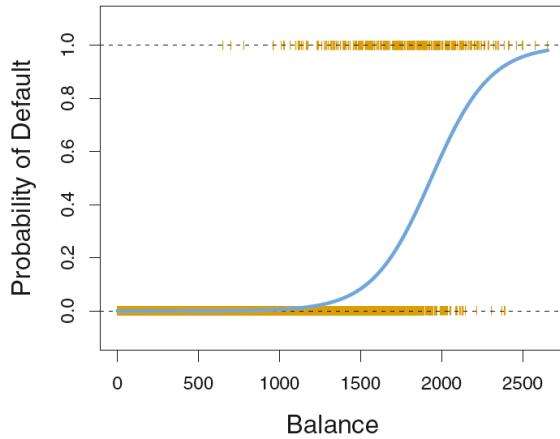


Рис. 5.8: Применение логистической регрессии в задаче оценки вероятности не вернуть задолженность.

Последнее свойство является весьма полезным. Например, в уже рассмотренной задаче при размере задолженности в районе 2000\$ оценка вероятности просрочки платежа сильно изменяется при увеличении или уменьшении задолженности на 100\$. С другой стороны при размере задолженности в 500\$ увеличение задолженности на 100\$ приводит только к незначительным изменениям требуемой оценки.

5.6.5. Оценка параметров

По функции $\pi(x)$ можно восстановить функцию g , которая фигурирует в определении обобщенной линейной модели:

$$\pi(x) \approx \frac{e^{\langle w, x \rangle}}{1 + e^{\langle w, x \rangle}} \iff \langle w, x \rangle \approx \underbrace{\ln \frac{\pi(x)}{1 - \pi(x)}}_{\text{Логит}}.$$

Риск

Отношение, стоящее под логарифмом, называется риском, а весь логарифм называется «логит». Именно поэтому метод называется логистической регрессией: логит приближается линейной комбинацией факторов.

Настройка модели происходит методом максимизации правдоподобия $L(X)$. Удобнее однако не максимизировать правдоподобие, а минимизировать минус логарифм от правдоподобия:

$$-\ln L(X) = -\sum_{i=1}^{\ell} \left(y_i \ln \pi(x_i) + (1 - y_i) \ln(1 - \pi(x_i)) \right)$$

Такой функционал также имеет названия log-loss, кросс-энтропия и другие.

Если изменить метку нулевого класса на -1 , то получится логистическая функция потерь в таком виде, в котором она встречалась в курсе до этого:

$$Q(w, X) = \sum_{i=1}^{\ell} \ln (1 + \exp(-y_i \langle w, x \rangle))$$

5.6.6. Решение задачи максимизации правдоподобия

Задача максимизации правдоподобия в логистической регрессии очень хорошо решается численно, поскольку правдоподобие — выпуклая функция, а следовательно, она имеет единственный глобальный максимум. Кроме того, ее градиент и гессиан могут быть хорошо оценены.

Если объекты из разных классов линейно разделимы в пространстве признаков, возникает проблема переобучения: сигмоида вырождается в «ступеньку».

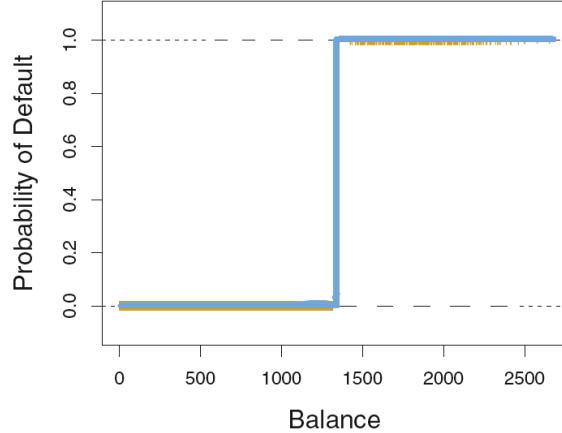


Рис. 5.9: Проблема переобучения в задаче логистической регрессии.

Например, такая ситуация возникает, если в уже упомянутой задаче оценки вероятности вернуть задолженность обучающая выборка такова, что все клиенты с задолженностью менее 1300\$ вернули платеж вовремя, а все клиенты с задолженностью более 1300\$ — нет.

В этом случае максимизация правдоподобия приводит к тому, что $\|w\| \rightarrow \infty$. В таких случаях необходимо использовать методы регуляризации, например L_1 или L_2 регуляризатор.

5.6.7. Предсказание отклика

Вероятности, которые дает логистическая регрессия, можно использовать для классификации, то есть для предсказания итоговых меток классов. Для этого выбирается порог p_0 и объект относится к классу 1 только в случае $\pi(x) > p_0$. В остальных случаях объект относится к классу 0.

Порог p_0 не следует выбирать всегда равным 0.5, как это может показаться из интуитивных соображений. Его необходимо подбирать для каждой задачи отдельно таким образом, чтобы обеспечить оптимальный баланс между точностью и полнотой классификатора.

Урок 6

Практические рекомендации по линейным моделям

Этот урок будет посвящен сложностям, с которыми можно столкнуться при применении линейных моделей к реальным задачам.

6.1. Масштабирование признаков

6.1.1. Пример: необходимость масштабирования

Понять необходимость масштабирования можно на следующем простом примере. Пусть необходимо найти минимум:

$$w_1^2 + w_2^2 \rightarrow \min_w.$$

Ответ в этом случае очевиден — это точка $(0, 0)$. При поиске этого минимума методом градиентного спуска с начальной точкой $w = (1, 1)$, вектор антиградиента будет иметь координаты $(-2, -2)$. Это вектор проходит через точку минимума, а значит при правильно подобранном размере шага, уже на первом шаге градиентного спуска можно попасть строго в точку минимума. Градиентный спуск будет хорошо работать на этой функции.

Изменим функцию:

$$w_1^2 + 100w_2^2 \rightarrow \min_w.$$

В этом случае линии уровня представляют собой эллипсы, сильно вытянутые вдоль оси x . Если запустить градиентный спуск из точки $w = (1, 1)$, вектор антиградиента в этой точке будет иметь координаты $(-2, -200)$. Вектор будет смотреть почти строго вниз и проходить мимо точки минимума функции. Более того, существует шанс на следующей итерации уйти еще дальше от минимума.

Итак, градиентный спуск работает хорошо, если линии уровня функции похожи на круги. В этом случае, откуда бы вы не начали, вектор градиента будет всегда смотреть в сторону минимума функции, а итеративный процесс будет сходиться довольно быстро. Если же линии уровня похожи на эллипсы, направление антиградиента будет слабо совпадать с направлением в сторону минимума функции. Градиентный спуск будет делать много лишних шагов. Сходимость будет медленная, и более того, существует риск расхождения итеративного процесса, если размер шага будет подобран неправильно.

6.1.2. Масштабирование выборки

Пусть требуется предсказать, будет ли выдан грант по заявке. Заявка характеризуется двумя признаками — сколько уже успешных заявок было у данного заявителя и год рождения заявителя. Масштабы этих двух признаков существенно отличаются: количество одобренных грантов обычно меряется единицами, а год рождения — тысячами. Из-за такого различия в масштабе линии уровня функции будут выглядеть скорее как эллипсы, а не как круги.

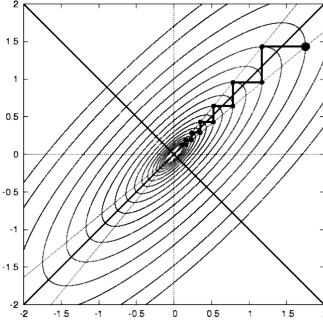


Рис. 6.1: Если масштабирование не было произведено, при использовании метода градиентного спуска будет сделано много лишних шагов, а также существует риск расхождения итеративного процесса.

В таком случае, чтобы без проблем пользоваться градиентным спуском, признаки необходимо привести к одному масштабу, то есть отмасштабировать. Будут рассматриваться два способа масштабирования.

Первый способ называется стандартизацией. Для начала необходимо вычислить вспомогательные величины: средние значения признаков и стандартные отклонения:

$$\mu_j = \frac{1}{\ell} \sum_{i=1}^{\ell} x_i^j, \quad \sigma_j = \sqrt{\frac{1}{\ell} \sum_{i=1}^{\ell} (x_i^j - \mu_j)^2}$$

Чтобы произвести стандартизацию признака, достаточно вычесть из него его среднее значение и разделить на его стандартное отклонение:

$$x_i^j := \frac{x_i^j - \mu_j}{\sigma_j}$$

После того, как это будет выполнено для всех признаков, сдвиги и различия в масштабах будут убраны.

Второй подход называется «масштабирование на отрезок $[0, 1]$ ». В этом случае также необходимо вычислить вспомогательные величины: максимальное и минимальное значение каждого признака на обучающей выборке:

$$m_j = \min(x_1^j, \dots, x_\ell^j), \quad M_j = \max(x_1^j, \dots, x_\ell^j).$$

После этого значение каждого признака на конкретном объекте преобразовывается следующим образом:

$$x_i^j := \frac{x_i^j - m_j}{M_j - m_j}.$$

Тогда минимальное значение признака отображается в ноль, а максимальное — в 1, то есть признаки масштабируются на отрезок $[0, 1]$.

6.2. Нелинейные зависимости

В этом разделе речь пойдет о спрямляющих пространствах, которые позволяют восстанавливать нелинейные зависимости с помощью линейных моделей. Для начала простой пример.

6.2.1. Нелинейная задача регрессии

Пусть решается задача регрессии с одним признаком, отложенным по оси x . По этому признаку требуется восстановить целевую переменную y .

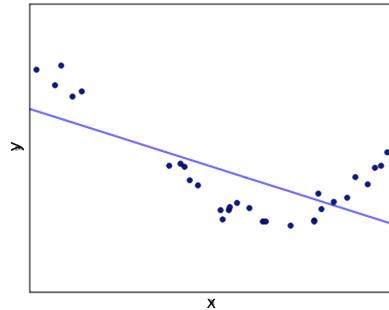


Рис. 6.2: Линейная модель сама по себе не способна восстанавливать нелинейные зависимости.

Как можно в этом убедиться, непосредственное применение линейной модели не дает желаемых результатов и плохо подходит для решения данной задачи. Зависимость y от x явно нелинейная.

Поэтому, кроме признака x , также рассматриваются признаки x^2 , x^3 и x^4 . Если построить линейную модель на этих признаках, в ней уже будет 5 коэффициентов.

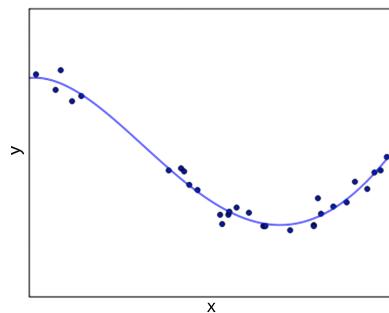


Рис. 6.3: Описать данные удалось с помощью перехода к другому признаковому пространству.

С помощью этой модели уже можно почти идеально описывать данные: она очень хорошо решает задачу, а также не переобучается.

Фактически был совершен переход к новому признаковому пространству из четырех признаков вместо одного, в котором была построена линейная модель. А на исходном пространстве эта модель уже нелинейная и отлично описывает данные.

6.2.2. Задача классификации с нелинейной границей

Другой пример — решается задача классификации с двумя признаками, которые отложены по осям.

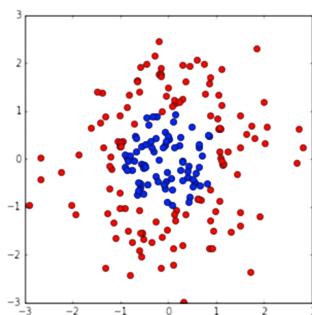


Рис. 6.4: Набор данных для рассматриваемой задачи классификации.

Видно, что разделяющая поверхность здесь не является линейной. Применение линейного классификатора дает следующий результат:

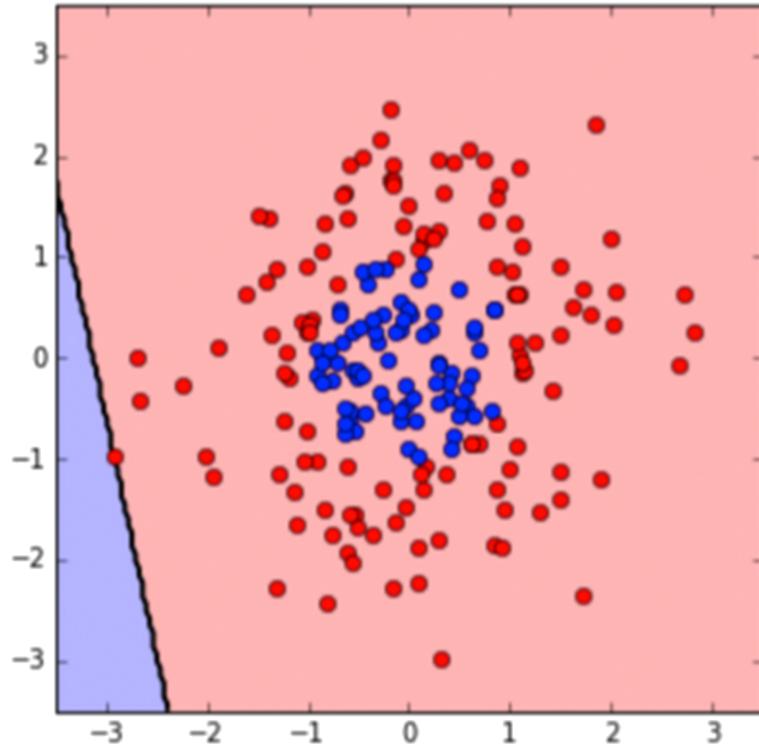


Рис. 6.5: Линейный классификатор не смог решить задачу.

То есть все объекты будут отнесены к красному классу: это лучшее, что он может сделать, но понятно, что это никуда не годится.

Но если кроме признаков x_1 и x_2 рассмотреть признаковое пространство, которое также включает в себя признаки x_1x_2 , x_1^2 и x_2^2 , результат будет совершенно иной: разделяющая поверхность будет иметь форму окружности и очень хорошо разделять красные и синие точки.

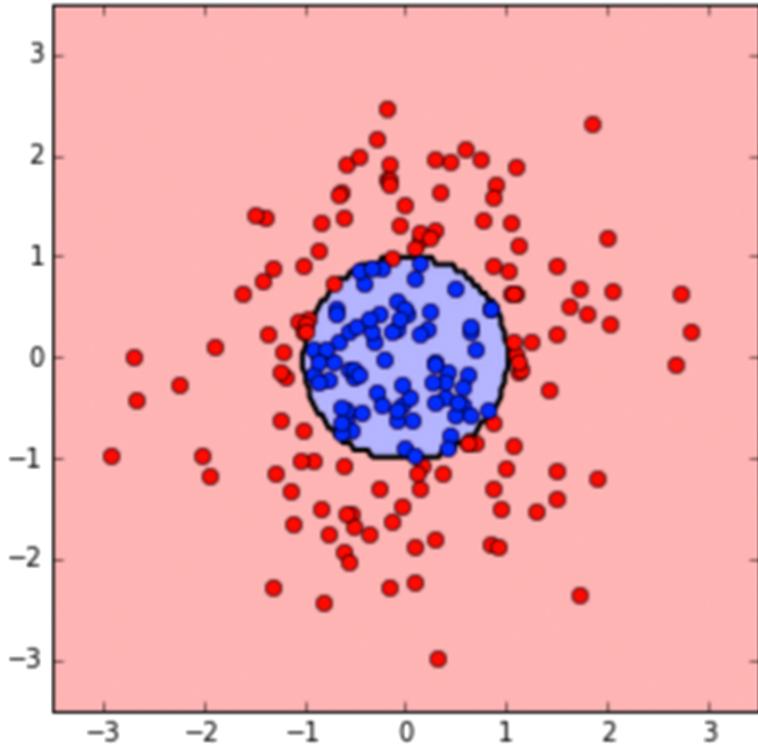


Рис. 6.6: Красные и синие точки хорошо разделились.

Здесь также в новом пространстве была построена линейная модель, которая является нелинейной в исходном признаковом пространстве.

6.2.3. Спрямляющее пространство

Два рассмотренных примера дают представление о спрямляющем пространстве. Спрямляющим пространством признаков называется такое пространство, в котором задача хорошо решается линейной моделью. Спрямляющее пространство может, в том числе, быть построено:

- Через добавление квадратичных признаков, то есть когда к исходным признакам добавляются их квадраты и попарные произведения:

$$(x_1, \dots, x_d) \rightarrow (x_1, \dots, x_d, x_1^2, \dots, x_d^2, x_1x_2, \dots, x_{d-1}x_d).$$

Следует особо отметить, что в таком случае число признаков увеличивается на порядок. Если объектов не слишком много, существует риск переобучения в таком большом признаковом пространстве.

- Через добавление полиномиальных признаков:

$$(x_1, \dots, x_d) \rightarrow (x_1, \dots, x_d, \dots, x_i x_j, \dots, x_i x_j x_k, \dots).$$

Этот способ следует использовать только, когда объектов достаточно много, то есть риск переобучения минимален, а также заранее известно, что зависимости очень нелинейные.

- Через логарифмирование (или любые другие нелинейные функции):

$$x_i \rightarrow \ln(x_i + 1), \quad x_i \rightarrow \ln(|x_i| + 1).$$

Ранее приводился пример про стоимость книг в интернет-магазине. В данном случае признаком будет стоимость книги, значение которого для большинства книг составит несколько сотен рублей. Но существуют и встречаются весьма часто очень дорогие книги, так что распределение этого признака будет иметь «тяжелый правый хвост». Известно, что линейные модели плохо применимы в таком случае и работают гораздо лучше, если распределение признаков близко к нормальному. Чтобы распределение признака «с хвостом» сделать более близким к нормальному, нужно прологарифмировать этот признак.

6.3. Работа с категориальными признаками

В этом разделе пойдет речь о том, как использовать категориальные признаки в линейных или других моделях.

Ранее уже было приведено множество примеров категориальных признаков:

- Город
- Цвет
- Тарифный план
- Марка автомобиля
- и так далее...

Особенность категориальных признаков состоит в том, что это элементы некоторого неупорядоченного множества, и нельзя говорить, что какое-то значение больше или меньше другого. Можно только сравнивать их на равенство. Но в линейных моделях нужно брать значение признака, умножать на вес, а потом складывать с другими числами, и эту операцию нельзя делать со значениями категориальных признаков. Категориальные признаки нужно сначала преобразовать, чтобы их можно было использовать в линейных моделях.

6.3.1. Бинарное кодирование

Один из наиболее популярных подходов к кодированию категориальных признаков — бинарное кодирование. Далее будут использоваться следующие обозначения. Пусть j -ый признак — категориальный и принимает n возможных значений:

$$c_1, \dots, c_n.$$

Пусть также $f_j(x)$ — значение этого признака на объекте x . Чтобы закодировать данный признак, вводятся n новых бинарных признаков:

$$b_1(x), \dots, b_n(x),$$

причем значение бинарного признака b_i равно единице только в том случае, если на данном объекте x значение категориального признака $f_j(x)$ равно c_i :

$$b_i(x) = [f_j(x) = c_i].$$

В результате один категориальный признак заменяется n бинарными признаками.

6.3.2. Бинарное кодирование (пример)

Пусть в качестве признака рассматривается цвет, причем он может принимать три значения: синий, зеленый или красный. Дано три объекта x_1, x_2, x_3 , на которых значения категориального признака:

$$f_j(x_1) = \text{синий}, \quad f_j(x_2) = \text{красный}, \quad f_j(x_3) = \text{синий}.$$

Поскольку категориальный признак принимает 3 значения, потребуется 3 бинарных признака, чтобы его закодировать. В результате получается следующая матрица (каждому объекту соответствует своя строка, а каждому столбцу — свое значение признака):

$$\begin{pmatrix} 1 & 0 & 0 \\ 0 & 0 & 1 \\ 1 & 0 & 0 \end{pmatrix}$$

6.3.3. Бинарное кодирование: новые значения

Часто возникает следующая проблема. При кодировании тестовой выборки может встретиться объект, на котором категориальный признак принимает новое ($n + 1$)-е значение, которое до этого не встречалось в обучающей выборке. В этом случае логичным подходом будет не добавлять новый признак (поскольку это сложно реализуется), а просто приравнять к нулю все существующие признаки. Действительно, по смыслу бинарных признаков (принимает ли категориальный признак значение c_i , $i \in \{1, 2, \dots, n\}$), каждый из них в таком случае должен равняться нулю.

6.4. Несбалансированные данные

В этом разделе пойдет речь о том, что такое несбалансированные выборки, к каким проблемам они могут привести, а также как бороться с такими проблемами.

6.4.1. Несбалансированная выборка

Пусть решается задача классификации с несбалансированной выборкой. Задача называется несбалансированной, если объектов одного класса существенно меньше, чем объектов остальных классов. Например, задача бинарной классификации называется несбалансированной, если объектов одного из двух классов менее 10%.

Примеров задач с несбалансированными выборками довольно много:

- Предсказание резких скачков курса доллара. Если определение резкого скачка подразумевает сильные изменения, то примеров таких изменений за всю историю — единицы, но при этом практически каждый день — отрицательный пример, то есть пример, когда такого скачка не было. Выборка в этом случае будет очень несбалансированной.
- Медицинская диагностика (больных, как правило, сильно меньше, чем здоровых)
- Обнаружение мошеннических транзакций (которых существенно меньше, чем обычных транзакций)
- Классификация текстов и так далее.

Основная проблема, связанная с несбалансированными выборками, состоит в том, что классификаторы минимизируют число неправильных ответов и никак не учитывают цены ошибок. Может возникнуть ситуация, когда выгоднее отнести все объекты к большему классу, не пытаясь как-то выделить объекты маленького класса. Другими словами, при работе с несбалансированными выборками классификаторы получаются очень плохие с точки зрения точности или полноты.

6.4.2. Undersampling

Первый подход к работе с несбалансированными выборками — undersampling. Его основная идея состоит в том, что часть объектов из большого класса выбрасываются из выборки. Пусть, например, есть три класса: очень крупный, крупный и небольшой

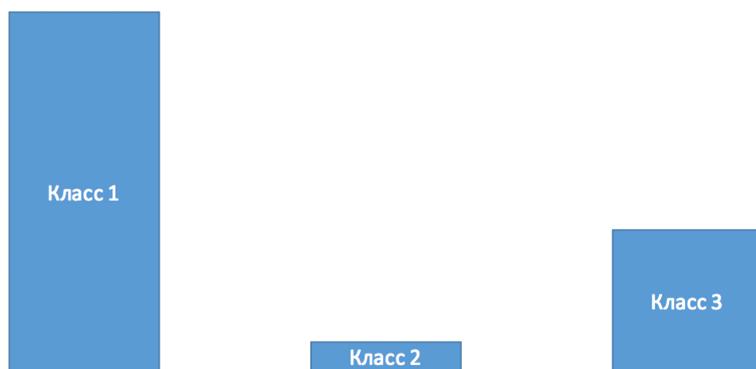


Рис. 6.7: Несбалансированная выборка

В этом случае необходимо выкинуть большую часть 1го класса и половину объектов 3го класса. Размеры классов примерно сравняются.

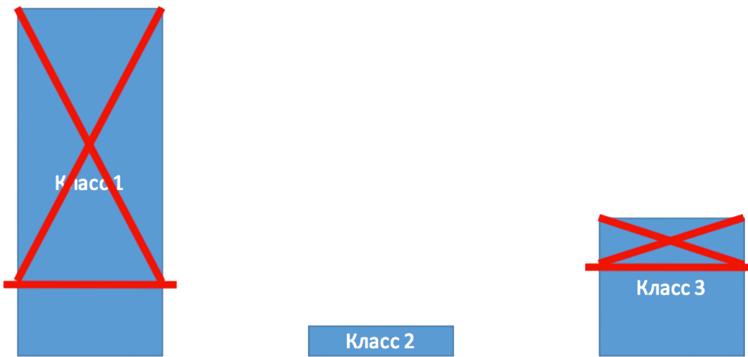


Рис. 6.8: Undersampling

При этом то, сколько именно объектов каждого класса выбрасывается — это гиперпараметр, который имеет смысл настраивать по отложенной выборке или по кросс-валидации.

6.4.3. Oversampling

Второй подход, oversampling, противоположен предыдущему: в данном случае объекты маленьких классов дублируются, чтобы выравнить соотношение классов.

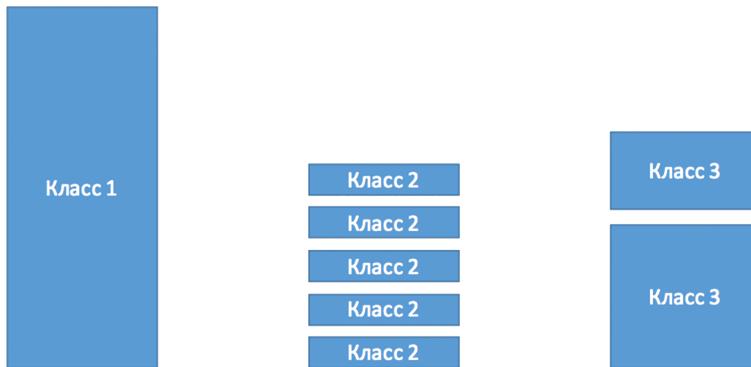


Рис. 6.9: Oversampling

В предыдущем примере 5 раз необходимо продублировать объекты 2го класса, а также продублировать случайную половину 3го класса. В этом случае размеры классов тоже выравниваются. То, на сколько будет увеличен каждый класс — тоже гиперпараметр.

Следует обратить внимание на одну особенность. Если задача решается на исходной выборке, то среднеквадратичная ошибка выглядит:

$$MSE(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2.$$

Но если делается oversampling, то есть дублируются какие-нибудь объекты, некоторые слагаемые будут входить в эту сумму несколько раз. Таким образом, вместо реального дублирования объектов, можно выставить соответствующие веса ν_i :

$$MSE(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} \nu_i (a(x_i) - y_i)^2.$$

6.4.4. Стратификация

Еще одна проблема, с которой можно столкнуться при работе с несбалансированными выборками, заключается в том, что при проведении кросс-валидации исходная выборка разбивается на k блоков примерно одинаковой длины. При этом, если выборка несбалансированная, может получиться ситуация, что в некоторые блоки объекты какого-то класса не попадут вообще. Такая ситуация крайне неприятна: при обучении на этом блоке получается классификатор, который никогда не видел один из классов.

Чтобы с этим бороться необходимо использовать стратификацию, то есть делать так, чтобы распределение классов в каждом блоке примерно совпадало с распределением классов в исходной выборке. В этом случае будет гарантироваться, что объекты каждого из классов будут представлены в каждом из блоков разбиения.

6.5. Многоклассовая классификация

В этом разделе рассказывается, как решать задачи многоклассовой классификации с помощью линейных моделей. Как следует из названия, в задачах многоклассовой классификации K возможных классов. Требуется научиться отличать каждый класс от всех остальных.

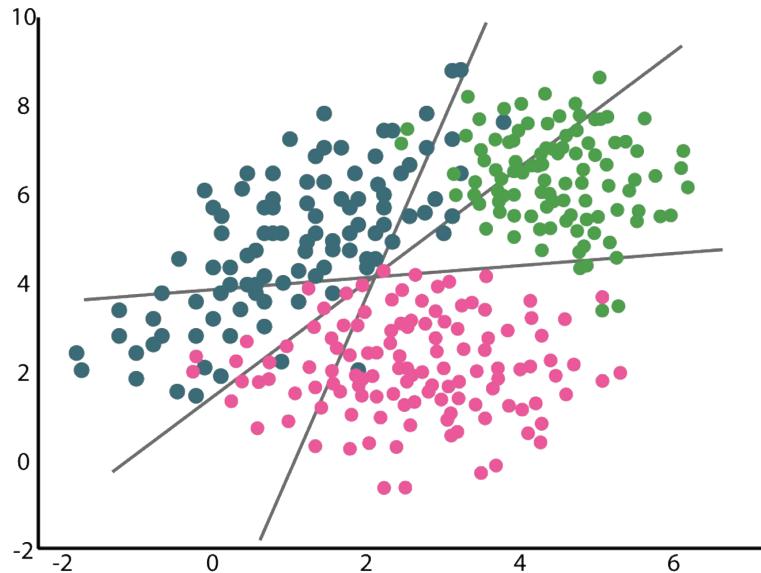


Рис. 6.10: Многоклассовая классификация

В случае бинарной классификации подход был простой — нужно было найти такой вектор весов w , что выражение $a(x) = \text{sign}\langle w, x \rangle$ определяло бы, какому классу этот объект относится.

6.5.1. ONE-VS-ALL

Этот метод можно применить к задаче многоклассовой классификации. Такой подход называется «один против всех». Как следует из названия, для каждого класса будет строиться свой бинарный классификатор. Задачей этого классификатора будет отделение данного класса от всех остальных.



Рис. 6.11: ONE-VS-ALL

Формально говоря, будут решаться K задач бинарной классификации. Для каждой (например k -ой) из этих задач будет весьма специфичная выборка:

$$X = (x_i, [y_i = k])_{i=1}^{\ell},$$

в которой объекты x_i остаются такими же, а ответы становятся бинарными. Будет построен некоторый линейный классификатор, который отделяет k -ый класс от всех остальных:

$$a_k(x) = \text{sign}\langle w_k, x \rangle.$$

Ранее уже было сказано, что уверенность классификатора в своем решении определяется значением скалярного произведения. Если знак скалярного произведения положительный, то чем больше его модуль, тем больше классификатор уверен в том, что данный конкретный объект относится к данному классу. Поэтому для построения многоклассового классификатора можно использовать следующий алгоритм:

$$a(x) = \text{argmax}_{k \in 1, \dots, K} \langle w_k, x \rangle,$$

который будет возвращать тот класс k , для которого уверенность соответствующего классификатора (то есть значение соответствующего скалярного произведения) больше всего.

6.5.2. Матрица ошибок

Для анализа того, насколько хорошо работает многоклассовый классификатор, удобно использовать матрицу ошибок. Каждая строка этой матрицы соответствует тем объектам, которые классификатор отнес к тому или иному классу, а каждый столбец соответствует объектам, которые на самом деле относятся к тому или иному классу.

Например, на пересечении первой строки и второго столбца стоит число q_{12} , которое показывает то, сколько объектов второго класса многоклассовый классификатор отнес к первому. Эта матрица позволяет понять, какие классы перепутываются чаще всего. Также можно измерять уже известные метрики качества, например:

$$\text{accuracy} = \frac{1}{\ell} \sum_{i=1}^{\ell} [a(x_i) = y_i].$$

Кроме того, можно считать точность и полноту (а также F -меру) для задачи отделения того или иного класса от всех остальных классов. Если точность и полнота будут таким образом вычислены для каждого из классов, они могут быть позднее усреднены, чтобы получить агрегированные оценки.

Урок 7

Решающие деревья

7.1. Решающие деревья

Решающие деревья — это семейство алгоритмов, которое очень сильно отличается от линейных моделей, но в то же время играет важную роль в машинном обучении.

7.1.1. Линейные модели (обзор)

До этого момента изучались линейные модели. К особенностям линейных моделей относится следующее:

- Линейные модели быстро учатся. В случае со среднеквадратичной ошибкой для вектора весов даже есть аналитическое решение. Также легко применять для линейных моделей градиентный спуск.
- При этом линейные модели могут восстанавливать только простые зависимости из-за ограниченного количества параметров (степеней свободы).
- В то же время линейные модели можно использовать для восстановления нелинейных зависимостей за счет перехода к спрямляющему пространству, что является довольно сложной операцией.

Отдельно стоит отметить, что линейные модели не отражают особенности процесса принятия решений у людей. На самом деле, когда человек хочет понять ту или иную вещь, он будет задавать последовательность из простых вопросов, которые в итоге приведут его к какому-нибудь ответу.

7.1.2. Решающие деревья (пример 1)

Чтобы понять принцип работы решающих деревьев, полезно рассмотреть следующий сильно упрощенный пример.

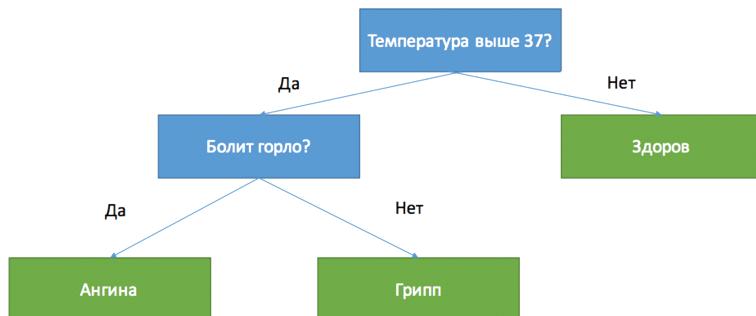


Рис. 7.1

Необходимо провести медицинскую диагностику. Врач, который проводит эту диагностику, знает только 2 заболевания — ангину и грипп. Поэтому сначала он спрашивает, какая температура у пациента. Если она меньше 37 градусов, он заключает, что пациент здоров, в ином случае — переходит к следующему вопросу, а

именно, спрашивает, болит ли у пациента горло. Если оно болит, врач ставит диагноз ангину, в ином случае — грипп.

Создатели курса не рекомендуют серьезно относиться к предложенному методу диагностики заболеваний.

7.1.3. Решающие деревья (пример 2)

Другой пример — для известной задачи определения того, выживет или не выживет тот или иной пассажир Титаника. Задача очень неплохо решается следующим решающим деревом:

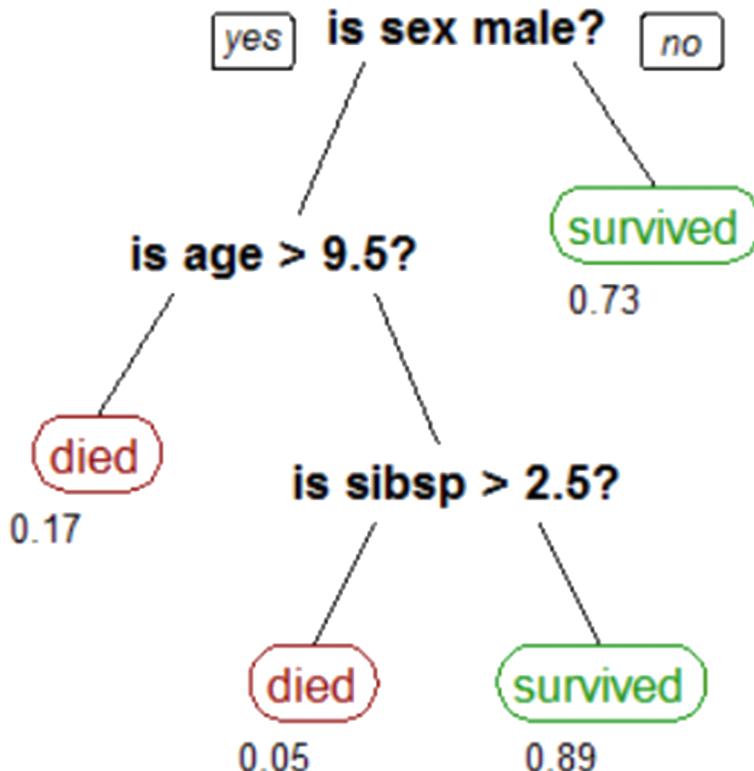


Рис. 7.2

В первую очередь спрашивается пол пассажира. Если это женщина, то решающее дерево сразу заявляет, что она выживает, и этот ответ верен в 73% случаев, и так далее.

7.1.4. Решающие деревья

Итак, были рассмотрены два примера решающих деревьев, которые представляли собой бинарные деревья, в каждой внутренней вершине записано условие, а в каждом листе дерева — прогноз. Строго говоря, не обязательно решающее дерево должно быть бинарным, но как правило используются именно бинарные.

Условия во внутренних вершинах выбираются крайне простыми. Наиболее частый вариант — проверить, лежит ли значение некоторого признака x^j левее, чем заданный порог t :

$$[x^j \leq t].$$

Это очень простое условие, которое зависит всего от одного признака, но его достаточно, чтобы решать многие сложные задачи.

Прогноз в листе является вещественным числом, если решается задача регрессии. Если же решается задача классификации, то в качестве прогноза выступает или класс, или распределение вероятностей классов.

7.1.5. Решающие деревья в задаче классификации

Пусть решается задача классификации с двумя признаками и тремя классами.

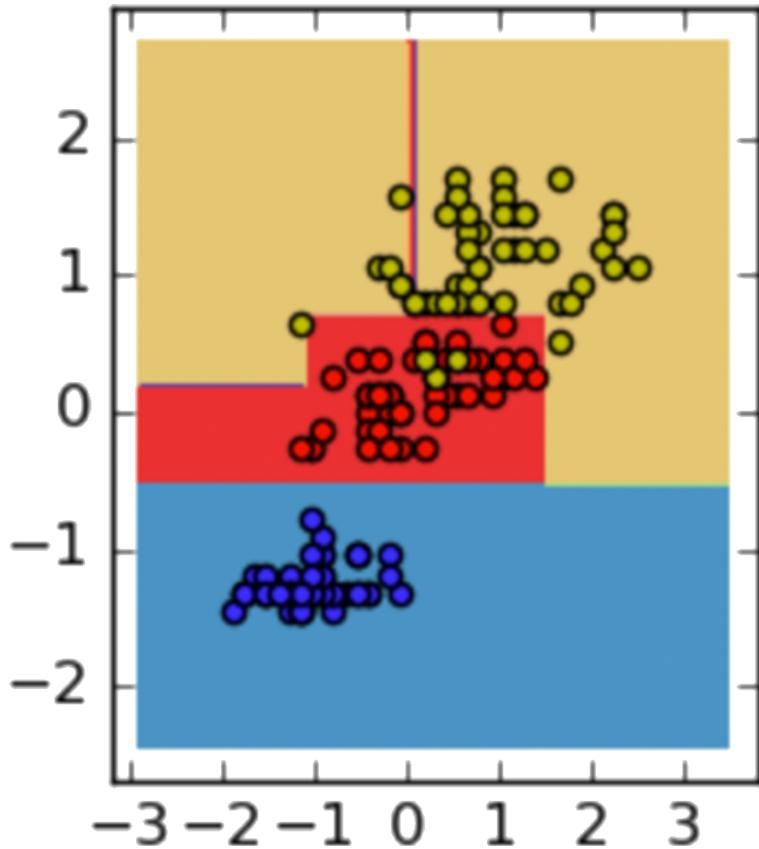


Рис. 7.3: Использование решающих деревьев в задачах классификации

Видно, что решающее дерево может очень неплохо отделить каждый класс от всех остальных. Видно, что разделяющая поверхность каждого класса кусочно-постоянная, и при этом каждая сторона поверхности параллельна оси координат, так как каждое условие сравнивает значение равно одного признака с порогом.

В то же время решающее дерево вполне может переобучиться: его можно сделать настолько глубоким, что каждый лист решающего дерева будет соответствовать ровно одному объекту обучающей выборки. В этом случае, если записать в каждом листе ответ соответствующего объекта, на обучающей выборке получается нулевая ошибка. Дерево получается явно переобученным. Вот пример такого дерева:

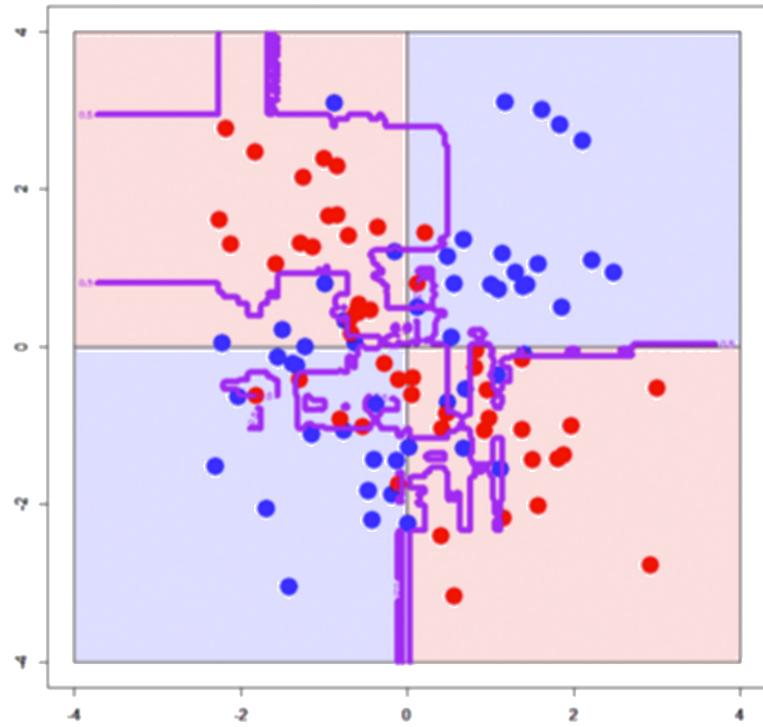


Рис. 7.4: Переобученное решающее дерево

Это дерево идеально отделило синий от красного класса, но разделяющая поверхность получилась безумно сложной — видно, что этот алгоритм переобучился и от него не будет никакой пользы на тестовой выборке.

7.1.6. Решающие деревья в задаче регрессии

Пусть решается задача регрессии с одним признаком, по которому нужно восстановить значение целевой переменной. Не очень глубокое дерево восстанавливает зависимость примерно так:

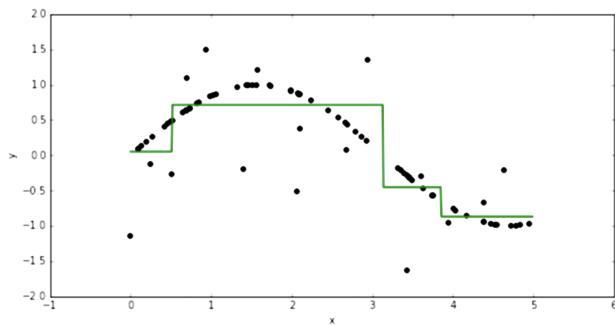


Рис. 7.5: Использование решающих деревьев в задачах регрессии

Восстановленная зависимость будет кусочно-постоянной, но в целом будет иметь неплохое качество. При увеличении глубины дерева получившаяся функция будет иметь следующий вид:

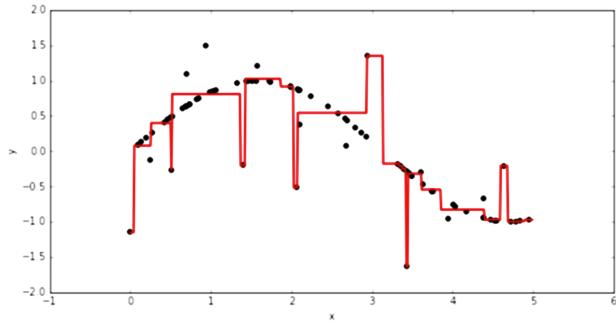


Рис. 7.6: Переобученное решающее дерево

Видно, что дерево подогналось под выбросы и его качество уже будет не таким хорошим. Дерево переобучилось из-за того, что его глубина слишком большая.

7.2. Обучение решающих деревьев

В данном разделе будет рассмотрен вопрос, как строить решающие деревья и как обучать их по конкретной выборке.

7.2.1. Переобучение деревьев

В предыдущем разделе было показано, что решающие деревья очень легко переобучаются. В том числе можно построить дерево, у которого каждый лист будет соответствовать одному объекту обучающей выборки.

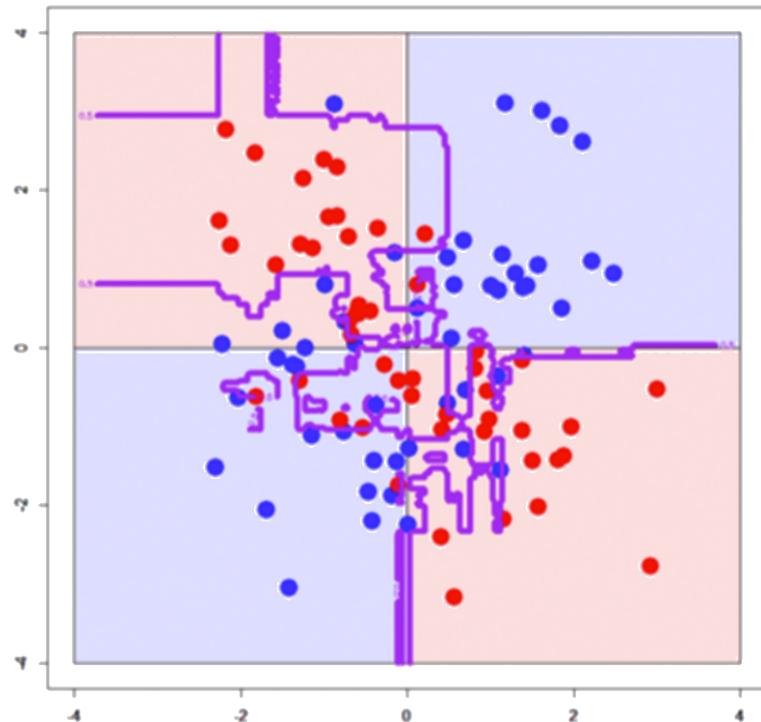


Рис. 7.7: Переобученное решающее дерево

Это дерево строит очень-очень сложную разделяющую поверхность и, очевидно, переобучено.

Поскольку всегда можно построить такое дерево, которое не ошибается на обучающей выборке и будет переобученным, имеет смысл искать минимальное (например, с минимальным числом листьев) дерево из имеющих нулевую ошибку. Но, к сожалению, задача отыскания такого дерева – NP-полная, то есть ее невозможно решить за разумное время.

7.2.2. Жадный способ построения

В машинном обучении применяется жадный способ построения решающего дерева от корня к листьям. Сначала выбирается корень, который разбивает выборку на две. Затем разбивается каждый из потомков этого корня и так далее. Дерево ветвится до тех пор, пока этого не будет достаточно.

Остается уточнить способ разбиения каждого потомка. Как было сказано ранее, в качестве условия в каждой вершине строящегося дерева будет использоваться простейшее условие: значение одного из признаков будет сравниваться с некоторым порогом.

Пусть в вершину t попало множество X_m объектов из обучающей выборки. Параметры в условии $[x^j \leq t]$ будут выбраны так, чтобы минимизировать данный критерий ошибки $Q(X_m, j, t)$, зависящий от этих параметров:

$$Q(X_m, j, t) \rightarrow \min_{j, t}.$$

Параметры j и t можно подбирать перебором. Действительно, признаков конечное число, а из всех возможных значений порога t можно рассматривать только те, при которых получаются различные разбиения. Можно показать, что таких значений параметра t столько, сколько различных значений признака x^j на обучающей выборке.

После того, как параметры были выбраны, множество X_m объектов из обучающей выборки разбивается на два множества

$$X_\ell = \{x \in X_m | [x^j \leq t]\}, \quad X_r = \{x \in X_m | [x^j > t]\},$$

каждое из которых соответствует своей дочерней вершине.

Предложенную процедуру можно продолжить для каждой из дочерних вершин: в этом случае дерево будет все больше и больше углубляться. Такой процесс рано или поздно должен остановиться, и очередная дочерняя вершина будет объявлена листком, а не разделена пополам. Этот момент определяется критерием остановки. Существует много различных вариантов критерия остановки:

- Если в вершину попал только один объект обучающей выборки или все объекты принадлежат одному классу (в задачах классификации), дальше разбивать не имеет смысла.
- Можно также останавливать разбиение, если глубина дерева достигла определенного значения.

Возможные критерии останова будут обсуждаться позднее в этом уроке.

Если какая-то вершина не была поделена, а была объявлена листом, нужно определить прогноз, который будет содергаться в данном листе. В этот лист попала некоторая подвыборка X_m исходной обучающей выборки и требуется выбрать такой прогноз, который будет оптимальен для данной подвыборки.

В задаче регрессии, если функционал – среднеквадратичная ошибка, оптимально давать средний ответ по этой подвыборке:

$$a_m = \frac{1}{|X_m|} \sum_{i \in X_m} y_i.$$

В задаче классификации оптимально возвращать тот класс, который наиболее популярен среди объектов в X_m :

$$a_m = \operatorname{argmax}_{y \in \mathbb{Y}} \sum_{i \in X_m} [y_i = y].$$

Если требуется указать вероятности классов, их можно указать как долю объектов разных классов в X_m :

$$a_{mk} = \frac{1}{|X_m|} \sum_{i \in X_m} [y_i = k].$$

7.3. Критерии информативности

В этом разделе речь пойдет о критериях информативности, с помощью которых можно выбирать оптимальное разбиение при построении решающего дерева.

7.3.1. Выбор критерия ошибки

Критерий ошибки записывается следующим образом:

$$Q(X_m, j, t) = \frac{|X_\ell|}{|X_m|} H(X_\ell) + \frac{|X_r|}{|X_m|} H(X_r)$$

и состоит из двух слагаемых, каждое из которых соответствует своему листу.

Функция $H(X)$ называется критерием информативности: ее значение должно быть тем меньше, чем меньше разброс ответов в X .

В случае регрессии разброс ответов — это дисперсия, поэтому критерий информативности в задачах регрессии записывается следующим образом:

$$H(X) = \frac{1}{|X|} \sum_{i \in X} (y_i - \bar{y}(X))^2, \quad \bar{y} = \frac{1}{|X|} \sum_{i \in X} y_i.$$

7.3.2. Критерий информативности Джини

Сформулировать критерий информативности для задачи классификации несколько сложнее. Пусть p_k — доля объектов класса k в выборке X :

$$p_k = \frac{1}{|X|} \sum_{i \in X} [y_i = k].$$

Критерий информативности Джини формулируется в терминах p_k :

$$H(X) = \sum_{k=1}^K p_k(1 - p_k).$$

Все слагаемые в сумме неотрицательные, поэтому критерий Джини также неотрицателен. Его оптимум достигается только в том случае, когда все объекты в X относятся к одному классу.

Одна из интерпретаций критерий Джини — это вероятность ошибки случайного классификатора. Классификатор устроен таким образом, что вероятность выдать класс k равна p_k .

7.3.3. Энтропийный критерий информативности

Еще один критерий информативности — энтропийный критерий:

$$H(X) = - \sum_{k=1}^K p_k \ln p_k.$$

В этом выражении полагается, что $0 \ln 0 = 0$.

Энтропийный критерий, как и критерий Джини, неотрицателен, а его оптимум также достигается только в том случае, когда все объекты в X относятся к одному классу.

Энтропийный критерий имеет интересный физический смысл. Он заключается в том, что показывает, насколько распределение классов в X отличается от вырожденного. Энтропия в случае вырожденного распределения равна 0: такое распределение характеризуется минимальной возможной степенью неожиданности. Напротив, равномерное распределение самое неожиданное, и ему соответствует максимальная энтропия.

7.4. Критерий останова и стрижка деревьев

В этом разделе речь пойдет о способах борьбы с переобучением деревьев, а именно о критериях останова и стрижке деревьев.

7.4.1. Критерий останова

Критерий останова используется, чтобы принять решение: разбивать вершину дальше или сделать листовой.

Худший случай решающего дерева — такое, в котором каждый лист соответствует своему объекту обучающей выборки. В этом случае дерево будет максимально переобученным и не будет общать информацию, полученную из обучающей выборки. Грамотно подобранный критерия останова позволяет бороться с переобучением.

Самый простой критерий останова проверяет, все ли объекты в вершине относятся к одному классу. Однако такой критерий останова может быть использован только в случае простых выборок, так как для сложных он останавливается только тогда, когда в каждом листе останется примерно по одному объекту.

Гораздо более устойчивый и полезный критерий проверяет, сколько объектов оказалось в вершине, и разбиение продолжается, если это число больше, чем некоторое выбранное n . Соответственно, если в вершину попало $\leq n$ объектов, она становится листовой. Параметр n нужно подбирать.

Случай $n = 1$ является худшим случаем, описанным выше. При этом выбирать n нужно так, чтобы по n объектам, которые попали в вершину, можно было устойчиво построить прогноз. Существует рекомендация, что n нужно брать равным 5.

Еще один критерий, гораздо более грубый, заключается в ограничении на глубину дерева. Этот критерий хорошо себя зарекомендовал при построении композиций, когда много решающих деревьев объединяют в один сложный алгоритм. Об этом пойдет речь позже.

7.4.2. Стрижка деревьев

Существует и другой подход к борьбе с переобучением деревьев — стрижка. Он заключается в том, что сначала строится решающее дерево максимальной сложности и глубины, до тех пор, пока в каждой вершине не окажется по 1 объекту обучающей выборки.

После этого начинается «стрижка», то есть удаление листьев в этом дереве по определенному критерию. Например, можно стричь до тех пор, пока улучшается качество некоторой отложенной выборки.

Существует мнение, и это подкреплено многими экспериментами, что стрижка работает гораздо лучше, чем простые критерии, о которых говорилось раньше. Но стрижка — очень ресурсоёмкая процедура, так как, например, может потребоваться вычисление качества дерева на некоторой валидационной выборке на каждом шаге.

На самом деле, сами по себе деревья на сегодняшний день почти не используются, они бывают нужны только для построения композиции и объединения большого числа деревьев в один алгоритм. В случае с композициями такие сложные подходы к борьбе с переобучением уже не нужны, так как достаточно простых критериев останова, ограничения на глубину дерева или на число объектов в листе.

7.5. Решающие деревья и категориальные признаки

В этом разделе будет рассказано, как использовать категориальные признаки в решающих деревьях.

До этого момента использовалось следующее условие в вершине каждого дерева:

$$[x^j \leq t].$$

Очевидно, что такое условие можно записывать только для вещественных или бинарных признаков.

7.5.1. N-арные деревья

Подход, который позволяет включить категориальные признаки в деревья, состоит в том, чтобы строить n -арные деревья, то есть такие деревья, что из каждой вершины могут выходить до n ребер. Пусть необходимо разбить некоторую вершину по некоторому признаку. Если этот признак — вещественный или бинарный, то все еще можно использовать простое условие с порогом t , поэтому интерес представляет именно случай категориального признака.

Если x^j — категориальный признак, который может принимать значения

$$\{c_1, \dots, c_n\},$$

можно разбить вершину на n вершин таким образом, что в i -ую дочернюю вершину идут объекты с $x^j = c_i$. Критерий ошибки такого разбиения строится по аналогии со случаем бинарного дерева. Поскольку X_n разбивается на n частей, а не на две, в выражении будет n слагаемых:

$$Q(X_m, j) = \sum_{i=1}^n \frac{|X_i|}{|X_m|} H(X_i) \rightarrow \min_j.$$

Таким образом, если вершину m нужно разбить, рассматриваются все возможные вещественные, бинарные и категориальные признаки. Для вещественных и бинарных признаков считается $Q(X_m, j, t)$, а для n -арных — так, как написано выше. Разбиение вершины будет происходить по тому признаку, для которого значение критерия ошибки будет минимальным.

Важное замечание состоит в том, что при делении вершины по категориальному признаку получается больше дочерних вершин, на которых, очень вероятно, будет достигаться более высокое качество и более низкое значение критерия информативности, поэтому, скорее всего, при таком подходе предпочтение почти всегда будет отдаваться разбиению по категориальным признакам с большим числом возможных значений. В результате получается много листьев в дереве, что почти гарантированно может привести к переобучению.

Однако это не всегда так. Если выборки настолько большие, что даже после разбиения по категориальному признаку в каждом поддереве будет оставаться много объектов, то такое дерево будет неплохо работать, поскольку оно будет восстанавливать сложные зависимости, и при этом не переобучаться при использовании должного критерия останова.

7.5.2. Бинарные деревья с разбиением множества значений

Другой подход позволяет не переходить к n -арным деревьям и продолжать работать с бинарными деревьями. Пусть также необходимо сделать разбиение вершины m , а категориальный признак x^j может принимать значения $C = \{c_1, \dots, c_n\}$.

Для этого сначала необходимо разбить множество значений категориального признака на два не пересекающихся подмножества:

$$C = C_1 \cup C_2, \quad C_1 \cap C_2 = \emptyset.$$

После того, как такое разбиение построено, условие в данной вершине будет выглядеть просто:

$$[x^j \in C_1]$$

Это условие проверяет, в какое из подмножеств попадает значение признака в данный момент на данном объекте. Главным вопросом остается то, как именно нужно разбивать множество C .

Полное количество возможных разбиений множества на два подмножества — 2^n . К счастью, есть хитрость, которая позволяет избежать полного перебора и, более того, работать с категориальным признаком как с вещественным. Для этого возможные значения категориального признака сортируются специальным образом $c_{(1)}, \dots, c_{(n)}$ и заменяются на натуральные числа $1, \dots, n$. После этого с данным признаком следует уже работать как с вещественным, а значение порога t будет определять разделение множества C на два подмножества.

Сортировать значения категориального признака в случае задачи бинарной классификации нужно по следующему принципу:

$$\frac{\sum_{i \in X_m} [x_i^j = c_{(1)}] [y_i = +1]}{\sum_{i \in X_m} [x_i^j = c_{(1)}]} \leq \dots \leq \frac{\sum_{i \in X_m} [x_i^j = c_{(n)}] [y_i = +1]}{\sum_{i \in X_m} [x_i^j = c_{(n)}]}$$

Фактически, значения категориального признака сортируются по возрастанию доли объектов $+1$ класса среди объектов выборки X_n с соответствующим значением этого признака.

Для задачи регрессии сортировка происходит похожим образом, но вычисляется не доля объектов положительного класса, а средний ответ по всем объектам, у которых значение категориального признака равно c :

$$\frac{\sum_{i \in X_m} [x_i^j = c_{(1)}] y_i}{\sum_{i \in X_m} [x_i^j = c_{(1)}]} \leq \dots \leq \frac{\sum_{i \in X_m} [x_i^j = c_{(n)}] y_i}{\sum_{i \in X_m} [x_i^j = c_{(n)}]}.$$

Главная особенность такого подхода состоит в том, что полученный результат полностью эквивалентен результату, который можно было бы получить в результате полного перебора. Это условие работает для критерия Джини, MSE и энтропийного критерия.

Урок 8

Случайные леса

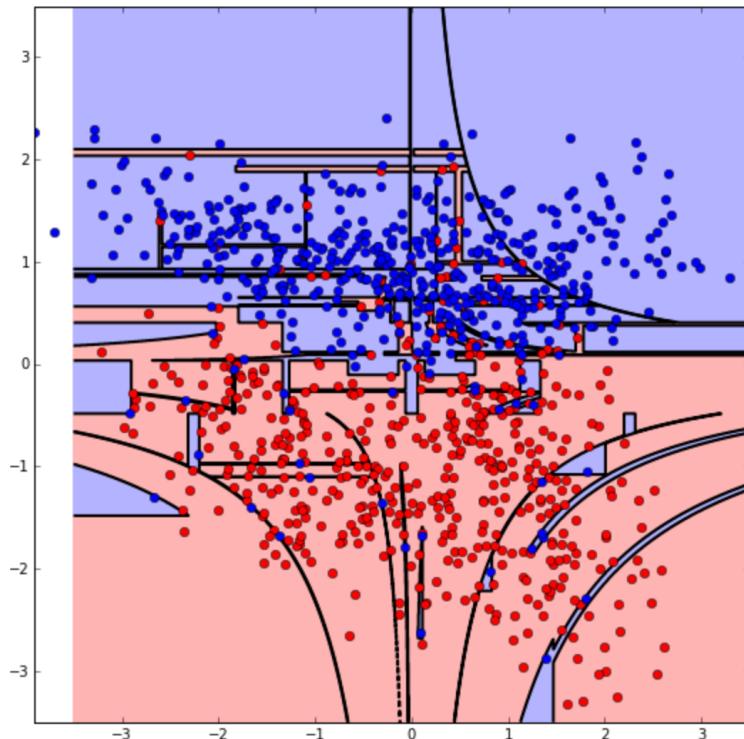
В прошлом уроке изучались решающие деревья и было установлено, что они способны восстанавливать очень сложные закономерности, следовательно, склонны к переобучению. Другими словами, деревья слишком легко подгоняются под обучающую выборку и получаются непригодными для построения прогнозов.

Но оказывается, решающие деревья очень хорошо подходят для объединения в композиции и построения одного непереобученного алгоритма на основе большого количества решающих деревьев.

8.1. Композиции деревьев

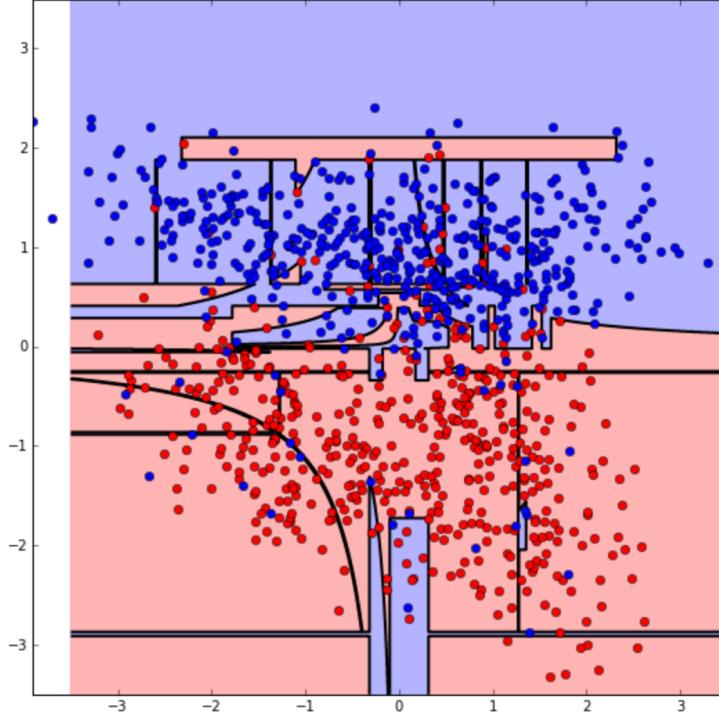
8.1.1. Основные недостатки решающих деревьев

Если взять сложную выборку и обучить на ней решающее дерево до конца, то есть пока в каждом из листьев не останется по одному объекту, получившаяся разделяющая поверхность будет очень сложной:



Даже если какой-то объект попадает в «гущу другого класса», разделяющая поверхность пытается «ловить» его и выдать на нем правильный ответ.

Если немного изменить обучающую выборку, например выкинуть пару объектов, то обученное на получившейся выборке дерево все еще будет характеризоваться изрезанной и переобученной разделяющей поверхностью, но совершенно другой:



Разделяющая поверхность крайне неустойчива к изменению выборки. Другими словами, решающее дерево обладает следующими серьезными недостатками:

- сильно переобучается
- сильно меняется при небольшом изменении выборки

На самом деле, второй пункт можно будет превратить в достоинство с помощью композиции.

8.1.2. Композиция алгоритмов

Композиция — это объединение N алгоритмов $b_1(x), \dots, b_N(x)$ в один. Идея заключается в том, чтобы обучить алгоритмы $b_1(x), \dots, b_N(x)$, а затем усреднить полученные от них ответы:

$$a(x) = \frac{1}{N} \sum_{n=1}^N b_n(x).$$

Это выражение непосредственно является ответом в задаче регрессии. В задачах классификации нужно будет взять знак от получившегося выражения:

$$a(x) = \text{sign} \frac{1}{N} \sum_{n=1}^N b_n(x),$$

Алгоритм $a(x)$, который возвращает среднее или знак среднего, называется композицией N алгоритмов $b_1(x), \dots, b_N(x)$, а они сами называются базовыми алгоритмами.

Например, пусть при решении задачи классификации с двумя классами использовались 6 базовых алгоритмов, которые на некотором объекте x выдали следующие ответы:

$$-1, -1, 1, -1, 1, -1.$$

Ответ композиции этих 6 алгоритмов будет:

$$a(x) = \text{sign} \left(-\frac{2}{6} \right) = -1.$$

Попросту говоря, объект был отнесен к классу -1 , так как за этот вариант «проголосовало» большинство базовых алгоритмов.

8.1.3. Рандомизация

Чтобы построить композицию, нужно сначала обучить N базовых алгоритмов, причем их нельзя обучать на всей обучающей выборке, так как в этом случае они получаются одинаковыми, и в их усреднении не будет никакого смысла.

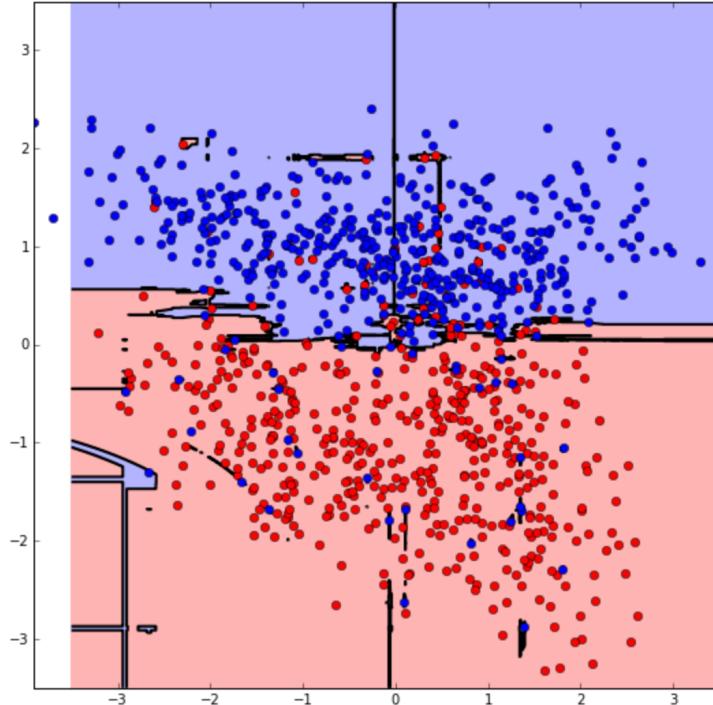
Использовать рандомизацию, то есть обучать базовые алгоритмы на разных подвыборках обучающей выборки, — это один из способов сделать базовые алгоритмы различными. А поскольку решающие деревья сильно меняются даже от небольших изменений обучающей выборки, такая рандомизация значительно повышает различность базовых алгоритмов.

Так называемый бутстррап — один из популярных подходов к построению подвыборок. Он заключается в том, что из обучающей выборки длины ℓ выбирают с возвращением ℓ объектов. При этом новая выборка также будет иметь размер ℓ , но некоторые объекты в ней будут повторяться, а некоторые объекты из исходной выборки в нее не попадут. Можно показать, что в бутстрэпированной выборке будет содержаться в среднем 63% различных объектов исходной выборки.

Другой подход к рандомизации — генерация случайного подмножества обучающей выборки. Размер этого случайного подмножества является гиперпараметром. Например, можно случайно взять половину исходной выборки и обучить на ней базовый алгоритм. Этот подход несколько проигрывает бутстррапу, так как содержит гиперпараметр, в то время как бутстррап без какой-либо настройки выдает подвыборку.

8.1.4. Композиция деревьев

Если с помощью бутстрэпа построить 100 базовых решающих деревьев и объединить их в композицию, разделяющая поверхность будет все еще сложная, но уже гораздо менее переобученная:



Разделяющая поверхность уже не подгоняется под большую часть попавших в гущу чужого класса объектов и в целом хорошо разделяет два класса. Увеличением количества базовых алгоритмов можно устранить оставшиеся погрешности.

8.2. Смещение и разброс

В этом разделе речь пойдет о разложении ошибки на шум, смещение и разброс. Эта техника позволяет глубже понять причины, почему усреднение алгоритмов позволяет повысить качество.

8.2.1. Разложение ошибки: шум, смещение и разброс

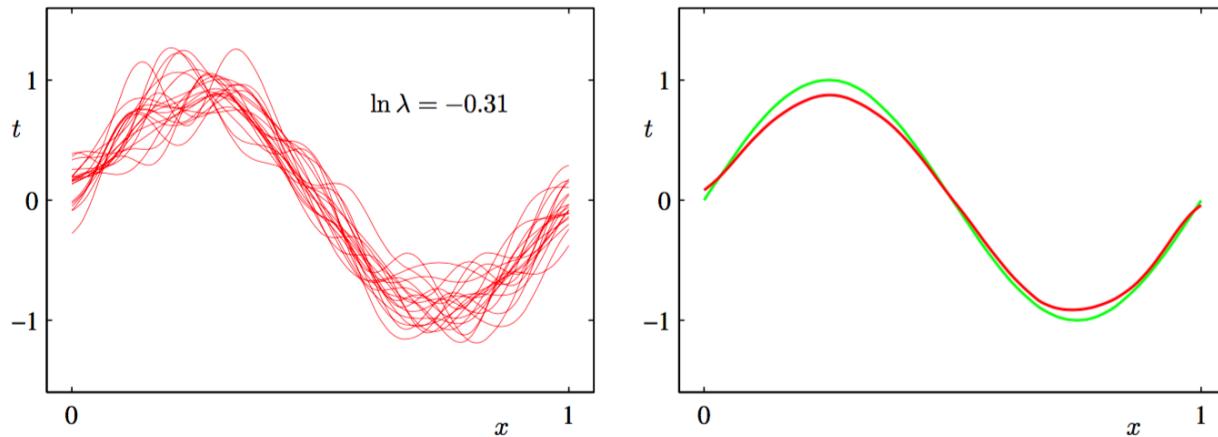
Ошибка алгоритма на новых тестовых данных складывается из трех компонент: шума, смещения и разброса. При этом все они характеризуют разные аспекты данных и модели, с помощью которой решается задача на этих данных:

- **Шум** — компонента ошибки алгоритма, которая будет проявляться даже на идеальной модели в этой задаче. Другими словами, шум является характеристикой данных и будет проявляться, какая бы модель не использовалась.

Пусть обучающая выборка генерируется из некоторого вероятностного распределения. На каждой конкретной обучающей выборке можно обучить некоторую модель и использовать обученную модель на тестовой выборке.

- **Смещение** — отклонение, усредненного по различным обучающим выборкам, прогноза заданной модели от прогноза идеальной модели.
- **Разброс** — дисперсия ответов моделей, обученных по различным обучающим выборкам. Разброс характеризует то, насколько сильно прогноз алгоритма зависит от конкретной обучающей выборки.

Продемонстрировать разложение ошибки на смещение и разброс можно на следующем примере. Рассматривается задача регрессии: требуется аппроксимировать истинную зависимость (изображена на правом графике зеленым) полиномом третьего порядка по обучающей выборке. Обучающая выборка представляет собой 10 случайных точек истинной зависимости, к которым был добавлен случайный шум. На левом графике изображены полиномы, получающиеся для различных обучающих выборок.



Усредненный полином (изображен красной линией на правом рисунке) практически идеально попадает в истинную зависимость, но каждый полином по отдельности существенно от нее отличается. Другими словами, используемое семейство алгоритмов обладает низким смещением, но довольно большим разбросом.

Линейные модели способны восстанавливать только линейные зависимости, а, следовательно, в случае нелинейных задач, которых подавляющее большинство, смещение при использовании таких алгоритмов будет большим. Разброс, наоборот, будет маленьким из-за малого числа параметров, сравнимого с количеством признаков. Вряд ли параметры линейной модели сильно поменяются при незначительном изменении обучающей выборки. Решающие деревья — полная противоположность. Они характеризуются низким смещением, то есть способны восстанавливать сложные закономерности, и большим разбросом: решающие деревья сильно меняются даже при небольших изменениях обучающей выборки.

8.2.2. Смещение и разброс композиции алгоритмов

При вычислении композиции базовых алгоритмов (с одинаковым смещением) смещение композиции совпадает со смещением отдельного базового алгоритма. Таким образом, поскольку деревья характеризуются низким

смещением, то же самое будет верно и для композиции деревьев. Следовательно, композиции деревьев тоже способны восстанавливать сложные закономерности.

Разброс композиции уже отличается от разброса одного базового алгоритма:

$$\left(\begin{array}{c} \text{разброс} \\ \text{композиции} \end{array} \right) = \frac{1}{N} \left(\begin{array}{c} \text{разброс одного} \\ \text{базового алгоритма} \end{array} \right) + \left(\begin{array}{c} \text{корреляция между} \\ \text{базовыми алгоритмами} \end{array} \right).$$

Если базовые алгоритмы независимы, то есть их прогнозы не коррелируют между собой, выражение упрощается:

$$\left(\begin{array}{c} \text{разброс} \\ \text{композиции} \end{array} \right) = \frac{1}{N} \left(\begin{array}{c} \text{разброс одного} \\ \text{базового алгоритма} \end{array} \right).$$

Фактически, композиция достаточного количества некоррелированных алгоритмов может дать идеальный алгоритм. Но, к сожалению, базовые алгоритмы всегда получаются в той или иной степени коррелированы, так как обучаются на подвыборках одной выборки. Таким образом, возникает необходимость уменьшения корреляции базовых алгоритмов.

8.2.3. Уменьшение корреляции базовых алгоритмов

Существуют следующие два подхода по уменьшению корреляции базовых алгоритмов:

1. **Бэггинг:** Обучение базовых алгоритмов происходит на случайных подвыборках обучающей выборки. Причем чем меньше размер случайной подвыборки, тем более независимыми получаются базовые алгоритмы.
2. **Метод случайных подпространств:** выбирается случайное подмножество признаков (столбцов матрицы «объекты–признаки») и очередной базовый алгоритм обучается только на этих признаках. Доля выбираемых признаков является гиперпараметром этого метода.

Два данных подхода — бэггинг и метод случайных подпространств — можно объединять и использовать одновременно.

8.3. Случайные леса

В этом разделе речь пойдет о случайных лесах, которые являются одним из лучших способов объединения деревьев в композиции.

8.3.1. Случайный лес

Ранее были получены следующие результаты:

- Ошибка может быть разложена на смещение и разброс.
- Смещение композиции близко к смещению одного базового алгоритма.
- Разброс при построении композиции уменьшается, причем тем сильнее, чем менее коррелированы базовые алгоритмы.

Рассмотренных в прошлый раз способов понижения корреляции между базовыми алгоритмами (бэггинг и метод случайных подпространств) оказывается недостаточно. Чтобы базовые алгоритмы были еще менее скоррелированными, имеет смысл сделать случайным их процесс построения.

8.3.2. Рандомизация процесса построения решающих деревьев

Процесс построения решающих деревьев представляет собой жадный алгоритм, работающий до выполнения критерия останова.

Пусть на некотором шаге алгоритма необходимо разбить вершину m , в которой оказалась выборка X_m , на две. В качестве условия разбиения используется сравнение j -го признака с порогом t :

$$[x^j \leq t].$$

Параметры j и t выбираются исходя из условия минимизации функции ошибки $Q(X_m, j, t)$:

$$Q(X_m, j, t) \rightarrow \min_{j,t} .$$

Рандомизировать процесс построения можно, если в задаче поиска оптимальных параметров выбирать j из случайного подмножества признаков размера q . Оказывается, что этот подход действительно позволяет сделать деревья менее коррелированными.

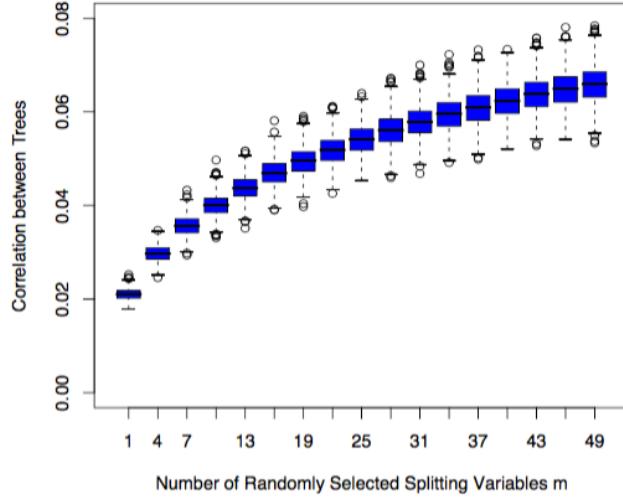


Рис. 8.1: Зависимость корреляции между деревьями от параметра q

По графику видно, что чем меньше «простор для выбора лучшего разбиения», то есть чем меньше q , тем меньше корреляции между получающимися решающими деревьями. Случай $q = 1$ соответствует абсолютно случайному выбору признака.

Для q есть некоторые рекомендации, которые неплохо работают на практике:

- В задаче регрессии имеет смысл брать $q = d/3$, то есть использовать треть от общего числа признаков.
- В задаче классификации имеет смысл брать $q = \sqrt{d}$.

8.3.3. Алгоритм построения случайноголеса

Чтобы построить случайный лес из N решающих деревьев, необходимо:

1. Построить с помощью бутстрата N случайных подвыборок \tilde{X}_n , $n = 1, \dots, N$.
2. Каждая получившаяся подвыборка \tilde{X}_n используется как обучающая выборка для построения соответствующего решающего дерева $b_n(x)$. Причем:
 - Дерево строится, пока в каждом листе окажется не более n_{min} объектов. Очень часто деревья строят до конца ($n_{min} = 1$), чтобы получить сложные и переобученные решающие деревья с низким смещением.
 - Процесс построения дерева рандомизирован: на этапе выбора оптимального признака, по которому будет происходить разбиение, он ищется не среди всего множества признаков, а среди случайногоподмножества размера q .
 - Следует обратить особое внимание, что случайное подмножество размера q выбирается заново каждый раз, когда необходимо разбить очередную вершину. В этом состоит основное отличие такого подхода от метода случайных подпространств, где случайное подмножество признаков выбиралось один раз перед построением базового алгоритма.
3. Построенные деревья объединяются в композицию:
 - В задачах регрессии $a(x) = \frac{1}{N} \sum_{n=1}^N b_n(x)$;
 - В задачах классификации $a(x) = \text{sign} \frac{1}{N} \sum_{n=1}^N b_n(x)$.

Одна из особенностей случайных лесов: они не переобучаются при росте числа базовых алгоритмов.

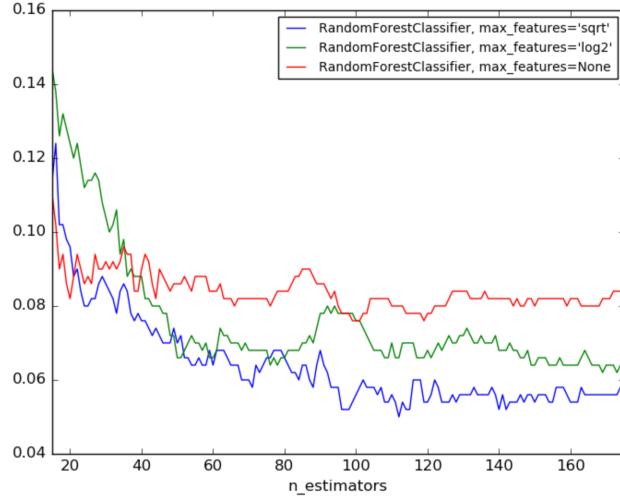


Рис. 8.2: Зависимость качества случайного леса от значения параметра q .

По графику видно, что ошибка на тесте сначала уменьшается с ростом числа базовых алгоритмов, а затем выходит на асимптоту. Не происходит роста ошибки при росте числа базовых алгоритмов.

8.4. Трюки со случайными лесами

Случайный лес обладает рядом интересных особенностей.

8.4.1. Возможность распараллеливания

Поскольку каждое дерево обучается независимо от всех остальных базовых решающих деревьев, его можно обучать на отдельном ядре или отдельном компьютере.

Фактически, данная задача допускает идеальное распараллеливание: скорость вычислений пропорциональна количеству задействованных вычислительных ядер.

8.4.2. Оценивание качества случайного леса

Каждое дерево из случайного леса обучается на бутстранированной выборке, в которую попадают приблизительно 63% объектов полной выборки. Таким образом, около 37% объектов выборки не использовались при обучении этого дерева, а значит их можно использовать для оценки обобщающей способности случайного леса.

Такой подход носит название out-of-bag и позволяет оценивать качество леса без использования отложенной выборки или кросс-валидации. Формула для оценки качества случайного леса из N деревьев в рамках подхода out-of-bag имеет вид:

$$OOB = \sum_{i=1}^{\ell} L \left(y_i, \frac{1}{\sum_{n=1}^N [x_i \notin X_n]} \sum_{n=1}^N [x_i \notin X_n] b_n(x_i) \right).$$

Эта формула устроена следующим образом. Для каждого объекта x_i из обучающей выборки вычисляется средний прогноз по тем деревьям, в обучающую выборку которых не входит объект x_i :

$$\frac{1}{\sum_{n=1}^N [x_i \notin X_n]} \sum_{n=1}^N [x_i \notin X_n] b_n(x_i).$$

Для полученного прогноза вычисляется значение ошибки. В качестве оценки качества случайного леса используется сумма таких значений для всех элементов выборки.

Также с помощью случайных лесов и out-of-bag можно отбирать наиболее важные признаки. Об этом пойдет речь в следующем курсе.

XGBoost

Евгений Соколов
sokolov.evg@gmail.com

5 сентября 2016 г.

Мы уже разобрались с двумя классами методов построения композиций — бустингом и бэггингом, и познакомились с градиентным бустингом и случайным лесом, которые являются наиболее яркими представителями этих классов. На практике реализация градиентного бустинга оказывается очень непростой задачей, в которой успех зависит от множества тонких моментов. В этом тексте мы рассмотрим конкретную реализацию градиентного бустинга — пакет XGBoost [1], который считается одним из лучших на сегодняшний день. Это подтверждается, например, активным его использованием в соревнованиях по анализу данных на kaggle.com.

1 Градиентный бустинг

Вспомним, что на каждой итерации градиентного бустинга вычисляется вектор сдвигов s , который показывает, как нужно скорректировать ответы композиции на обучающей выборке, чтобы как можно сильнее уменьшить ошибку:

$$s = \left(-\frac{\partial L}{\partial z} \Big|_{z=a_{N-1}(x_i)} \right)_{i=1}^{\ell} = -\nabla_s \sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + s_i)$$

После этого новый базовый алгоритм обучается путем минимизации среднеквадратичного отклонения от вектора сдвигов s :

$$b_N(x) = \arg \min_{b \in \mathcal{A}} \sum_{i=1}^{\ell} (b(x_i) - s_i)^2$$

Также можно вычислять шаг с помощью методов второго порядка. В этом случае задача поиска базового алгоритма примет вид

$$b_N(x) = \arg \min_{b \in \mathcal{A}} \sum_{i=1}^{\ell} \left(b(x_i) - \frac{s_i}{h_i} \right)^2, \quad (1.1)$$

где через h_i мы обозначили вторые производные функции потерь:

$$h_i = \frac{\partial^2 L}{\partial z^2} \Big|_{z=a_{N-1}(x_i)}$$

1.0.1 Альтернативный подход

Мы аргументировали использование среднеквадратичной функции потерь тем, что она наиболее проста для оптимизации. Попробуем найти более адекватное обоснование этому выбору.

Мы хотим найти алгоритм $b(x)$, решающий следующую задачу:

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + b(x_i)) \rightarrow \min_b$$

Разложим функцию L в каждом слагаемом в ряд Тейлора до второго члена с центром в ответе композиции $a_{N-1}(x_i)$:

$$\begin{aligned} \sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + b(x_i)) &\approx \\ &\approx \sum_{i=1}^{\ell} \left(L(y_i, a_{N-1}(x_i)) - s_i b(x_i) + \frac{1}{2} h_i b^2(x_i) \right) \end{aligned}$$

Первое слагаемое не зависит от нового базового алгоритма, и поэтому его можно выкинуть. Получаем функционал

$$\sum_{i=1}^{\ell} \left(-s_i b(x_i) + \frac{1}{2} h_i b^2(x_i) \right) \quad (1.2)$$

Покажем, что он очень похож на среднеквадратичный из формулы (1.1). Преобразуем его:

$$\begin{aligned} &\sum_{i=1}^{\ell} \left(b(x_i) - \frac{s_i}{h_i} \right)^2 \\ &= \sum_{i=1}^{\ell} \left(b^2(x_i) - 2 \frac{s_i}{h_i} b(x_i) + \frac{s_i^2}{h_i^2} \right) = \{ \text{последнее слагаемое не зависит от } b \} \\ &= \sum_{i=1}^{\ell} \left(b^2(x_i) - 2 \frac{s_i}{h_i} b(x_i) \right) \\ &= \sum_{i=1}^{\ell} \frac{2}{h_i} \left(-s_i b(x_i) + \frac{1}{2} h_i b^2(x_i) \right) \end{aligned}$$

Видно, что последняя формула совпадает с (1.2) с точностью до коэффициентов $\frac{2}{h_i}$. Можно считать, что в (1.2) они отброшены для большей устойчивости функционала — из-за них может произойти деление на ноль в окрестности оптимума.

2 Регуляризация

Будем далее работать с функционалом (1.2). Он измеряет лишь ошибку композиции после добавления нового алгоритма, никак при этом не штрафуя за излишнюю сложность модели. Ранее мы решали проблему переобучения путем ограничения глубины деревьев, можно подойти к вопросу и более гибко. Мы выясняли, что

дерево $b(x)$ можно описать формулой

$$b(x) = \sum_{j=1}^J b_j [x \in R_j]$$

Его сложность зависит от двух показателей:

1. Число листьев J . Чем больше листьев имеет дерево, тем сложнее его разделяющая поверхность, тем больше у него параметров и тем выше риск переобучения.
2. Норма коэффициентов в листьях $\|b\|^2 = \sum_{j=1}^J b_j^2$. Чем сильнее коэффициенты отличаются от нуля, тем сильнее данный базовый алгоритм будет влиять на итоговый ответ композиции.

Добавляя регуляризаторы, штрафующие за оба этих вида сложности, получаем следующую задачу:

$$\sum_{i=1}^{\ell} \left(-s_i b(x_i) + \frac{1}{2} h_i b^2(x_i) \right) + \lambda J + \frac{\mu}{2} \sum_{j=1}^J b_j^2 \rightarrow \min_b$$

Если вспомнить, что дерево $b(x)$ дает одинаковые ответы на объектах, попадающих в один лист, то можно упростить функционал:

$$\sum_{j=1}^J \left\{ \underbrace{\left(- \sum_{i \in R_j} s_i \right)}_{=-S_j} b_j + \frac{1}{2} \underbrace{\left(\mu + \sum_{i \in R_j} h_i \right)}_{=H_j} b_j^2 + \lambda \right\}$$

Каждое слагаемое здесь можно минимизировать по b_j независимо. Заметим, что отдельное слагаемое представляет собой параболу относительно b_j , благодаря чему можно аналитически найти оптимальные коэффициенты в листьях:

$$b_j = \frac{S_j}{H_j + \mu}$$

Подставляя данное выражение обратно в функционал, получаем, что ошибка дерева с оптимальными коэффициентами в листьях вычисляется по формуле

$$H(b) = \frac{1}{2} \sum_{j=1}^J \frac{S_j^2}{H_j + \mu} + \lambda J \tag{2.1}$$

3 Обучение решающего дерева

Мы получили функционал $H(b)$, который для заданной структуры дерева вычисляет минимальную ошибку, которую можно получить путем подбора коэффициентов в листьях. Заметим, что он прекрасно подходит на роль критерия информативности — с его помощью можно принимать решение, какое разбиение вершины

является наилучшим! Значит, с его помощью мы можем строить дерево. Будем выбирать разбиение так, чтобы оно решало следующую задачу максимизации:

$$Q = H(b_l) + H(b_r) - H(b) - \lambda \rightarrow \max$$

За счет этого мы будем выбирать структуру дерева так, чтобы оно как можно лучше решало задачу минимизации исходной функции потерь. При этом введем вполне логичный критерий останова: вершину нужно объявить листом, если даже лучшее из разбиений приводит к отрицательному значению функционала Q .

4 Заключение

Итак, градиентный бустинг в XGBoost имеет ряд важных особенностей.

1. Базовый алгоритм приближает направление, посчитанное с учетом вторых производных функции потерь.
2. Отклонение направления, построенного базовым алгоритмом, измеряется с помощью модифицированного функционала — из него удалено деление на вторую производную, за счет чего избегаются численные проблемы.
3. Функционал регуляризуется — добавляются штрафы за количество листьев и за норму коэффициентов.
4. При построении дерева используется критерий информативности, зависящий от оптимального вектора сдвига.
5. Критерий останова при обучении дерева также зависит от оптимального сдвига.

Список литературы

- [1] Tianqi Chen, Carlos Guestrin (2016). XGBoost: A Scalable Tree Boosting System. // <http://arxiv.org/abs/1603.02754>

Урок 9

Градиентный бустинг

9.1. Композиции простых алгоритмов

В начале данного урока обсудим, почему случайные леса подходят не для всех задач.

9.1.1. Недостатки случайного леса

Случайный лес — композиция глубоких деревьев, которые строятся независимо друг от друга. Но такой подход имеет следующую проблему. Обучение глубоких деревьев требует очень много вычислительных ресурсов, особенно в случае большой выборки или большого числа признаков.

Если ограничить глубину решающих деревьев в случайном лесе, то они уже не смогут улавливать сложные закономерности в данных. Это приведет к тому, что сдвиг будет слишком большим.

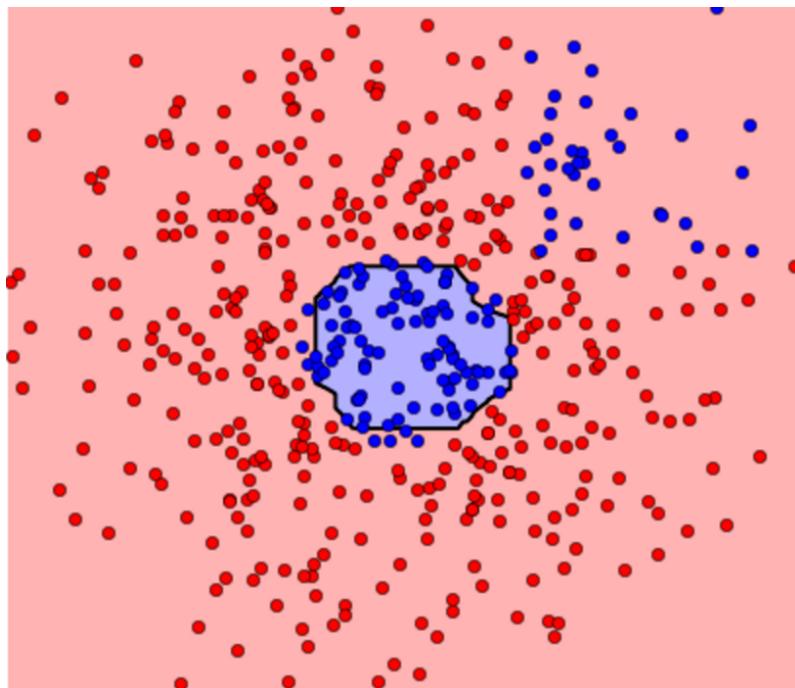


Рис. 9.1: Неглубокие деревья не способны улавливать все закономерности в данных. В данном случае синий класс состоит из двух групп объектов, но неглубокое дерево смогло уловить только центральную группу. На объектах из второй группы такое дерево ошибается.

Вторая проблема со случайнм лесом состоит в том, что процесс построения деревьев является ненаправленным: каждое следующее дерево в композиции никак не зависит от предыдущих. Из-за этого для решения сложных задач необходимо огромное количество деревьев.

9.1.2. Бустинг: основная идея

Решить данные проблемы можно с помощью так называемого бустинга. Бустинг — это подход к построению композиций, в рамках которого:

- Базовые алгоритмы строятся последовательно, один за другим.
- Каждый следующий алгоритм строится таким образом, чтобы исправлять ошибки уже построенной композиции.

Благодаря тому, что построение композиций в бустинге является направленным, достаточно использовать простые базовые алгоритмы, например неглубокие деревья.

9.1.3. Бустинг на примере задачи регрессии

Пусть дана задача регрессии, в которой в качестве ошибки используется среднеквадратичная ошибка:

$$MSE(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2.$$

Для начала необходимо обучить первый простой алгоритм (например, неглубокое решающее дерево):

$$b_1(x) = \operatorname{argmin}_b \frac{1}{\ell} \sum_{i=1}^{\ell} (b(x_i) - y_i)^2.$$

Такая задача минимизации квадратичной ошибки легко решается, например, градиентным спуском. Этот алгоритм будет первым алгоритмом в строящейся композиции.

Второй алгоритм должен быть обучен таким образом, чтобы композиция первого и второго алгоритмов:

$$b_1(x_i) + b_2(x_i)$$

имела наименьшую из возможных ошибку на обучающей выборке:

$$b_2(x) = \operatorname{argmin}_b \frac{1}{\ell} \sum_{i=1}^{\ell} (b_1(x_i) + b(x_i) - y_i)^2 = \operatorname{argmin}_b \frac{1}{\ell} \sum_{i=1}^{\ell} (b(x_i) - (y_i - b_1(x_i)))^2.$$

Другими словами, алгоритм $b_2(x)$ улучшает качество работы алгоритма $b_1(x)$. Продолжая по аналогии на N шаге очередной алгоритм $b_N(X)$ будет определяться следующим образом:

$$b_N(x) = \operatorname{argmin}_b \frac{1}{\ell} \sum_{i=1}^{\ell} \left(b(x_i) - \left(y_i - \sum_{n=1}^{N-1} b_n(x_i) \right) \right)^2.$$

Процесс продолжается до тех пор, пока ошибка композиции $b_1(x) + \dots + b_N(x)$ не будет устраивать.

9.2. Градиентный бустинг

Градиентный бустинг является одним из лучших способов направленного построения композиции на сегодняшний день. В градиентном бустинге строящаяся композиция

$$a_N(x) = \sum_{n=1}^N b_n(x)$$

является суммой, а не их усреднением базовых алгоритмов $b_i(x)$. Это связано с тем, что алгоритмы обучаются последовательно и каждый следующий корректирует ошибки предыдущих.

Пусть задана функция потерь $L(y, z)$, где y — истинный ответ, z — прогноз алгоритма на некотором объекте. Примерами возможных функций потерь являются:

- среднеквадратичная ошибка (в задаче регрессии):

$$L(y, z) = (y - z)^2$$

- логистическая функция потерь (в задаче классификации):

$$L(y, z) = \log(1 + \exp(-yz)).$$

9.2.1. Инициализация

В начале построения композиции по методу градиентного бустинга нужно ее инициализировать, то есть построить первый базовый алгоритм $b_0(x)$. Этот алгоритм не должен быть сколько-нибудь сложным и не стоит тратить на него много усилий. Например, можно использовать:

- алгоритм $b_0(x) = 0$, который всегда возвращает ноль (в задаче регрессии);
- более сложный $b_0(x) = \frac{1}{\ell} \sum_{i=1}^{\ell} y_i$, который возвращает средний по всем элементам обучающей выборки истинный ответ (в задаче регрессии);
- алгоритм $b_0(x) = \operatorname{argmax}_{y \in \mathbb{Y}} \sum_{i=1}^{\ell} [y_i = y]$, который всегда возвращает метку самого распространенного класса в обучающей выборке (в задаче классификации).

9.2.2. Обучение базовых алгоритмов

Обучение базовых алгоритмов происходит последовательно. Пусть к некоторому моменту обучены $N - 1$ алгоритмов $b_1(x), \dots, b_{N-1}(x)$, то есть композиция имеет вид:

$$a_{N-1}(x) = \sum_{n=1}^{N-1} b_n(x).$$

Теперь к текущей композиции добавляется еще один алгоритм $b_N(x)$. Этот алгоритм обучается так, чтобы как можно сильнее уменьшить ошибку композиции на обучающей выборке:

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + b(x_i)) \rightarrow \min_b.$$

Сначала имеет смысл решить более простую задачу: определить, какие значения s_1, \dots, s_ℓ должен принимать алгоритм $b_N(x_i) = s_i$ на объектах обучающей выборки, чтобы ошибка на обучающей выборке была минимальной:

$$F(s) = \sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + s_i) \rightarrow \min_s,$$

где $s = (s_1, \dots, s_\ell)$ — вектор сдвигов.

Другими словами, необходимо найти такой вектор сдвигов s , который будет минимизировать функцию $F(s)$. Поскольку направление наискорейшего убывания функции задается направлением антиградиента, его можно принять в качестве вектора s :

$$s = -\nabla F = \begin{pmatrix} -L'_z(y_1, a_{N-1}(x_1)), \\ \vdots \\ -L'_z(y_\ell, a_{N-1}(x_\ell)) \end{pmatrix}.$$

Компоненты вектора сдвигов s , фактически, являются теми значениями, которые на объектах обучающей выборки должен принимать новый алгоритм $b_N(x)$, чтобы минимизировать ошибку строящейся композиции.

Обучение $b_N(x)$, таким образом, представляет собой задачу обучения на размеченных данных, в которой $\{(x_i, s_i)\}_{i=1}^{\ell}$ — обучающая выборка, и используется, например, квадратичная функция ошибки:

$$b_N(x) = \operatorname{argmin}_b \frac{1}{\ell} \sum_{i=1}^{\ell} (b(x_i) - s_i)^2.$$

Следует обратить особое внимание на то, что информация об исходной функции потерь $L(y, z)$, которая не обязательно является квадратичной, содержится в выражении для вектора оптимального сдвига s . Поэтому для большинства задач при обучении $b_N(x)$ можно использовать квадратичную функцию потерь.

9.3. Описание алгоритма градиентного бустинга

1. **Инициализация:** инициализация композиции $a_0(x) = b_0(x)$, то есть построение простого алгоритма b_0 .
2. **Шаг итерации:**

- (a) Вычисляется вектор сдвига

$$s = -\nabla F = \begin{pmatrix} -L'_z(y_1, a_{n-1}(x_1)), \\ \dots \\ -L'_z(y_\ell, a_{n-1}(x_\ell)) \end{pmatrix}.$$

- (b) Строится алгоритм

$$b_n(x) = \operatorname{argmin}_b \frac{1}{\ell} \sum_{i=1}^{\ell} (b(x_i) - s_i)^2,$$

параметры которого подбираются таким образом, что его значения на элементах обучающей выборки были как можно ближе к вычисленному вектору оптимального сдвига s .

- (c) Алгоритм $b_n(x)$ добавляется в композицию

$$a_n(x) = \sum_{m=1}^n b_m(x)$$

3. Если не выполнен критерий останова (об этом будет рассказано далее), то выполнить еще один шаг итерации. Если критерий останова выполнен, остановить итерационный процесс.

9.4. Проблема переобучения градиентного бустинга

9.4.1. Проблема переобучения градиентного бустинга

На следующем графике изображена зависимость ошибки градиентного бустинга от числа используемых деревьев на обучающей и контрольной выборках.

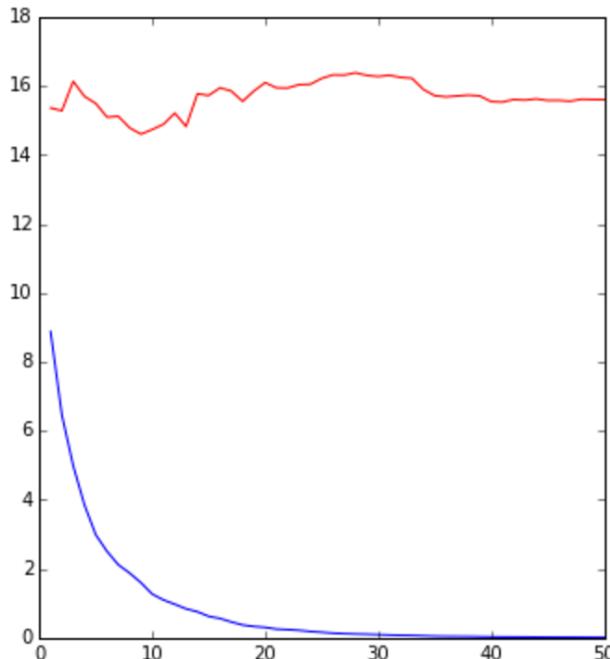


Рис. 9.2: Ошибка в зависимости от числа деревьев: синяя линия — ошибка на обучающей выборке. Красная линия — ошибка на контрольной выборке.

По мере увеличения числа деревьев ошибка на обучающей выборке постепенно уходит в 0. Ошибка на контрольной выборке существенно больше ошибки на обучающей выборке, достигает минимума примерно на 10 итераций, а затем начинает опять возрастать. Имеет место переобучение.

Это связано с тем, что базовый алгоритм пытается приблизить вектор антиградиента на обучающей выборке. Но в градиентном бустинге используются очень простые базовые алгоритмы, например невысокие решающие деревья, которые не могут хорошо аппроксимировать вектор антиградиента на обучающей выборке. Вектор, построенный алгоритмом, будет указывать не в сторону наискорейшего убывания ошибки, то есть вместо градиентного спуска можно получить случайное блуждание. Из-за этого и получается не очень хорошее качество на контроле.

9.4.2. Сокращение размера шага

Чтобы решить эту проблему, нужно «не доверять» направлению, которое построил базовый алгоритм и лишь чуть-чуть смешаться в сторону этого вектора:

$$a_N(x) = a_{N-1}(x) + \eta b_N(x),$$

где $\eta \in (0, 1]$ — длина шага. Это обеспечивает очень аккуратное движение в пространстве, что делает возможным нахождение локального минимума.

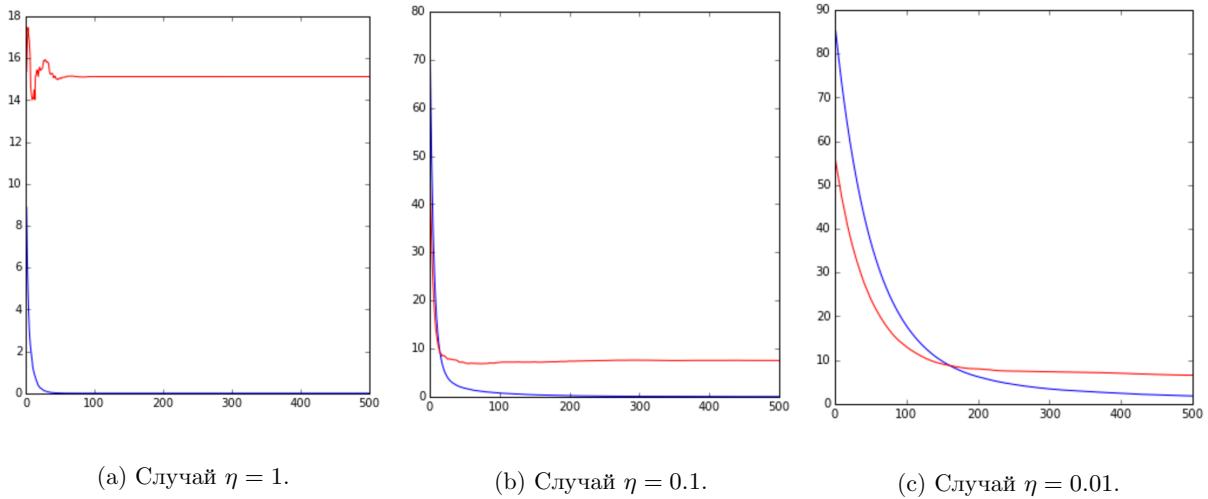


Рис. 9.3: Качество градиентного бустинга на обучающей выборке и контроле при различных η .

Как видно по графикам, при $\eta = 0.1$ качество на контрольной выборке уже существенно лучше, то есть в некотором смысле удалось побороть переобучение. При еще меньшей длине шага $\eta = 0.01$ градиентному бустингу требуется существенно больше итераций, чтобы достичь чуть-чуть большего качества. Таким образом:

- Чем меньше размер шага, тем больше нужно базовых алгоритмов, чтобы достичь хорошего качества, и тем больше времени занимает процесс.
- Чем меньше размер шага, тем лучшего качества можно достичь.

Другими словами, приходится выбирать: или быстро получить достаточно хорошее качество, или получить качество чуть-чуть лучше за большее время.

9.4.3. Подбор гиперпараметров

Размер шага η также является гиперпараметром градиентного бустинга, как и число итераций N . Эти гиперпараметры следует подбирать либо по отложенной выборке, либо по кросс-валидации. Но подбирать сразу два гиперпараметра довольно сложно, поэтому обычно используют одну из двух следующих стратегий:

1. Зафиксировать размер шага η и подбирать число итераций N
2. Зафиксировать число итераций N и подбирать размер шага η

Обе стратегии неплохо работают, но если время и ресурсы не ограничены, можно попробовать подобрать оба параметра одновременно.

9.4.4. Стохастический градиентный бустинг

Бэггинг — еще один подход к борьбе с переобучением градиентного бустинга, который заключается в том, что каждый базовый алгоритм обучается не на всей выборке, а на некоторой ее случайной подвыборке. Такой подход еще называется стохастическим градиентным бустингом.

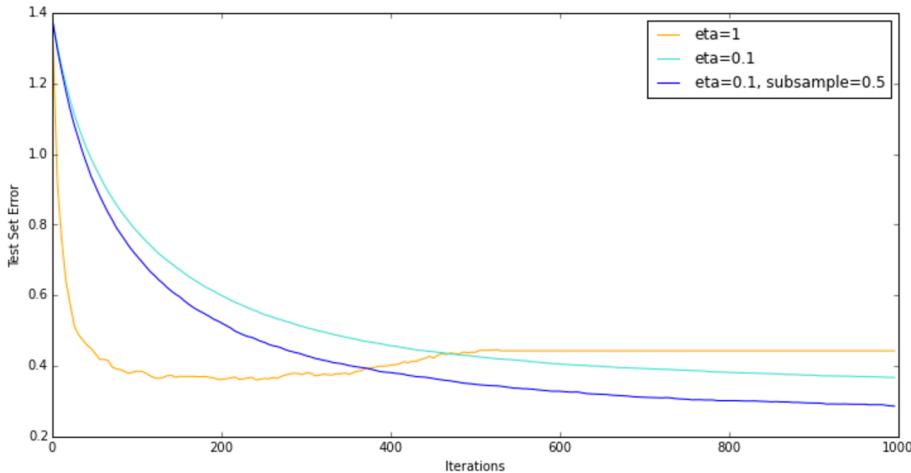


Рис. 9.4: Качество на контрольной выборке в зависимости от числа деревьев.

На графике изображены зависимость качества на контроле от числа деревьев:

- **Оранжевая кривая:** без применения сокращения шага и без бэггинга. Кривая обладает явно выраженным минимумом, после чего алгоритм начинает переобучаться.
- **Бирюзовая кривая:** при использовании сокращения шага с параметром $\eta = 0.1$ без бэггинга. В этом случае требуется больше итераций, чтобы достичь хорошего качества, но и при этом достигается более низкое значение ошибки. То есть метод сокращения шага действительно работает.
- **Синяя кривая:** при использовании сокращения шага с параметром $\eta = 0.1$ и бэггинга с размером подвыборки равным половине обучающей выборки. Форма кривой совпадает с формой бирюзовой кривой, но при этом за такое же количество итераций алгоритм достигает более низкой ошибки.

То есть использование стохастического градиентного бустинга позволяет уменьшить ошибку или достичь такой же ошибки при таком же числе итераций.

9.5. Градиентный бустинг для регрессии и классификации

9.5.1. Градиентный бустинг

В градиентном бустинге в качестве базовых алгоритмов, как правило, используются не очень глубокие (глубина выбирается от 2 до 8, обычно ближе к 2) решающие деревья. Использования таких не очень глубоких деревьев все же достаточно, чтобы восстанавливать сложные закономерности, поскольку бустинг строит направленную композицию. Чтобы решить возникающую в градиентном бустинге проблему с переобучением, необходимо использовать оба подхода борьбы с ним: сокращение шага и бэггинг.

9.5.2. Градиентный бустинг для регрессии

Типичный функционал ошибки в регрессии — это среднеквадратичная ошибка:

$$MSE(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} (a(x_i) - y_i)^2.$$

При этом функция потерь, которая измеряет ошибку для одного объекта:

$$L(y, z) = (z - y)^2, \quad L'_z(y, z) = 2(z - y),$$

где z — это прогноз нашего алгоритма, а y — истинный ответ на данном объекте. Соответственно, вектор сдвигов, каждая компонента которого показывает, как нужно модифицировать ответ на каждом объекте обучающей выборки, чтобы уменьшить среднеквадратичную ошибку, имеет вид:

$$s = \begin{pmatrix} -2(a_{N-1}(x_1) - y_1) \\ \dots \\ -2(a_{N-1}(x_\ell) - y_\ell) \end{pmatrix}.$$

9.5.3. Градиентный бустинг для классификации

В задаче бинарной классификации ($\mathbb{Y} = \{-1, +1\}$) популярным выбором для функции потерь является логистическая функция потерь:

$$\sum_{i=1}^n \log(1 + \exp(-y_i a(x_i))),$$

где $a(x) \in \mathbb{R}$ — оценка принадлежности положительному классу. Если $a(x) > 0$, классификатор относит объект x к классу $+1$, а при $a(x) \leq 0$ — к классу -1 . Причем, чем больше $|a(x)|$, тем больше классификатор уверен в своем выборе. Функция потерь в этом случае записывается следующим образом:

$$L(y, z) = \log(1 + \exp(-yz)), \quad L'_z(y, z) = -\frac{y}{1 + \exp(yz)}.$$

Вектор сдвигов s в этом случае будет иметь вид:

$$s = \begin{pmatrix} \frac{y_1}{1 + \exp(y_1 a_{N-1}(x_1))} \\ \dots \\ \frac{y_\ell}{1 + \exp(y_\ell a_{N-1}(x_\ell))} \end{pmatrix}.$$

Новый базовый алгоритм будет настраиваться таким образом, чтобы вектор его ответов на объектах обучающей выборки был как можно ближе к s . После того, как вычислен алгоритм $a_N(x)$, можно оценить вероятности принадлежности объекта x к каждому из классов:

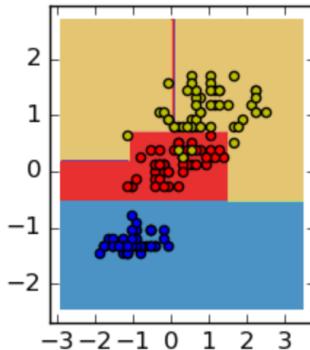
$$P(y = 1|x) = \frac{1}{1 + \exp(-a_N(x))}, \quad P(y = -1|x) = \frac{1}{1 + \exp(a_N(x))}.$$

9.6. Градиентный бустинг для решающих деревьев

В этом разделе речь пойдет о том, как применять градиентный бустинг, если базовый алгоритм — это решающие деревья.

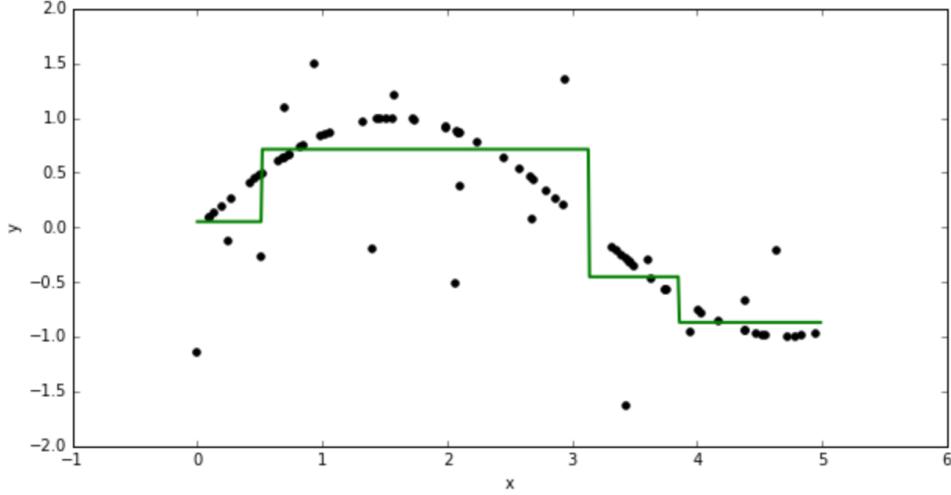
9.6.1. Поверхности, которые восстанавливают решающие деревья

В задаче классификации с тремя классами решающее дерево может разделить выборку примерно вот так:



Поскольку условия в каждой вершине дерева есть сравнение значения какого-то признака с порогом, решающее дерево выделяет каждый класс с помощью некой области, стороны которой параллельны осям координат.

В задаче регрессии функция, которую восстанавливает решающее дерево — кусочно постоянная:



В данном случае есть четыре участка, на каждом из которых функция возвращает постоянное значение.

Решающее дерево, таким образом, разбивает все пространство объектов на J областей:

$$R_1, R_2, \dots, R_J,$$

в каждой из которых дерево возвращает постоянное предсказание. Пусть b_j — предсказание дерева в области R_j , тогда решающее дерево $b(x)$ можно записать в следующем виде:

$$b(x) = \sum_{j=1}^J [x \in R_j] b_j.$$

Здесь $[x \in R_j]$ — индикатор того, что объект x попал в область R_j .

9.6.2. Градиентный бустинг для решающих деревьев

В градиентном бустинге каждый новый базовый алгоритм b_N прибавляется к уже построенной композиции:

$$a_N(x) = a_{N-1}(x) + b_N(x)$$

Если базовые алгоритмы — это решающие деревья

$$b_N(x) = \sum_{j=1}^J [x \in R_{Nj}] b_{Nj},$$

тогда новая композиция a_N будет выглядеть следующим образом:

$$a_N(x) = a_{N-1}(x) + \sum_{j=1}^J [x \in R_{Nj}] b_{Nj}$$

Последнее выражение можно проинтерпретировать не только как прибавление одного решающего дерева, но и как прибавление J очень простых алгоритмов, каждый из которых возвращает постоянное значение в некоторой области и ноль во всем остальном пространстве. Можно подобрать каждый прогноз b_{Nj} , где N — номер дерева, j — номер листа в этом дереве, таким образом, чтобы он был оптимальным с точки зрения исходной функции потерь:

$$\sum_{i=1}^{\ell} L \left(y_i, a_{N-1}(x) + \sum_{j=1}^J [x \in R_{Nj}] b_{Nj} \right) \rightarrow \min_{b_1, \dots, b_J}$$

Можно показать, что данная задача распадается на J подзадач:

$$b_{Nj} = \operatorname{argmin}_{\gamma \in R} \sum_{x_i \in \mathbb{R}_j} L(y_i, a_{N-1}(x) + \gamma).$$

Такая задача часто решается аналитически или любым простым методом. Итак, структура базового решающего дерева (структура областей R_j) в градиентном бустинге настраивается минимизацией среднеквадратичной ошибки. Потом можно переподобрать ответы в листьях, то есть перенастроить их, так, чтобы они были оптимальны не с точки зрения среднеквадратичной ошибки (с помощью которой строилось дерево), а с точки зрения исходной функции потерь L . Это позволяет существенно увеличить скорость сходимости градиентного бустинга.

Например, в задаче классификации с логистической функцией потерь, практически оптимальные значения для прогнозов в листьях имеют вид:

$$b_{Nj} = -\frac{\sum_{x_i \in \mathbb{R}_j} s_i}{\sum_{x_i \in \mathbb{R}_j} |s_i|(1 - |s_i|)}.$$

Урок 10

Нейронные сети

10.1. Однослойная нейронная сеть

В этом уроке речь пойдет о нейронных сетях. Нейронная сеть — это универсальная модель, решающая широкий класс задач регрессии и классификации.

10.1.1. Нейронная сеть и задача регрессии

Пусть $\{(x_i, y_i)\}_{i=1}^{\ell}$ — обучающая выборка в задаче регрессии, $x_i \in \mathbb{R}^d$ — признаковое описание i -го объекта выборки, $y_i \in \mathbb{R}^1$ — значение целевой зависимой переменной на i -ом объекте выборки. Требуется построить поверхность $a(x)$, аппроксимирующую неизвестную зависимость.

Задача оценки риска в страховании и финансах — пример задачи регрессии. Признаками в данной задаче являются время до страхового случая x_1 и цена страховки x_2 . Требуется оценить ожидаемый риск y . Поверхность $a(x)$, которая аппроксимирует исторический риск y в точках x , представлена на следующем графике.

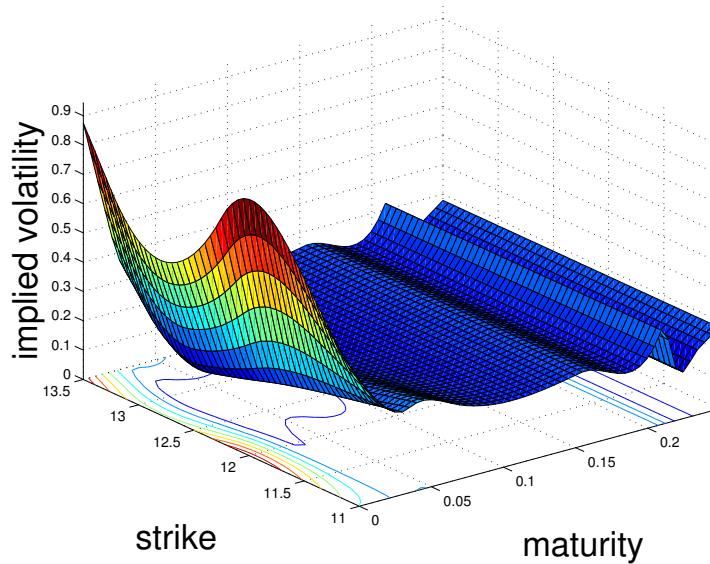


Рис. 10.1: Аппроксимирующая поверхность в задаче оценки риска.

Другие примеры задач регрессии $a(x) \mapsto y$:

- x — вектор исторических цен потребления электроэнергии, y — цена электроэнергии в следующий час;
- x — векторизованный снимок поверхности земли со спутника, y — объем зеленых насаждений;
- x — история продаж товара, y — уровень потребительского спроса.

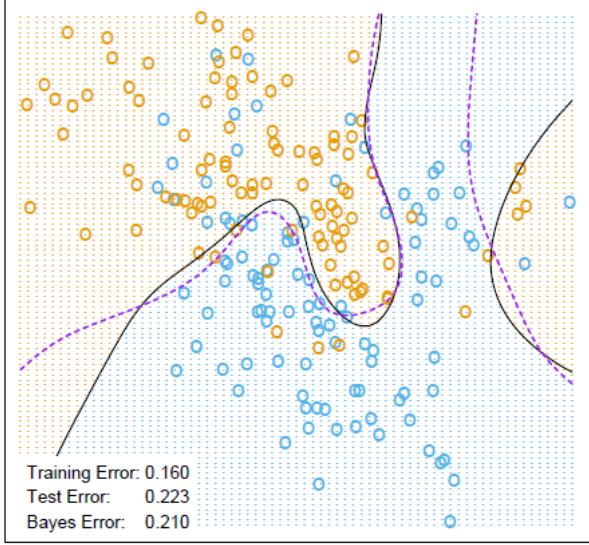


Рис. 10.2: Задача классификации с двумя признаками x_1 и x_2 (отложены по осям). На графике изображена разделяющая поверхность, которая отделяет объекты синего класса от объектов оранжевого класса.

10.1.2. Нейронная сеть и задача классификации

В задачах классификации целевая переменная y принимает конечное множество значений и называется меткой класса. Пусть $\{(x_i, y_i)\}_{i=0}^\ell$ — обучающая выборка в задаче бинарной классификации, $x_i \in \mathbb{R}^d$ и $y_i \in \{1, -1\}$ — признаковое описание и метка класса для i -го объекта соответственно. Требуется построить разделяющую поверхность

$$a(x) \mapsto y \in \{-1, 1\},$$

которая отделяет объекты одного класса от объектов другого класса таким образом, что, если $a(x) > 0$, то объект x принадлежит классу $y = +1$, и, если $a(x) \leq 0$, то x принадлежит классу $y = -1$.

Другие примеры задач классификации $a(x) \mapsto y$:

- x — временной ряд акселерометра мобильного телефона, y — вид физической активности;
- x — страница документа, y — релевантность этого документа по поисковому запросу;
- x — нотная запись в музыкальном произведении, y — это следующая нота, которая предполагается к проигрыванию.

10.1.3. Нейронная сеть как универсальная модель

Нейронная сеть считается универсальной моделью, так как она способна аппроксимировать любые поверхности. Соответствующую теорему сформулировал в 1957 году Андрей Николаевич Колмогоров.

Теорема (А. Н. Колмогоров, 1957) Каждая непрерывная функция $a(x)$, заданная на единичном кубе d -мерного пространства, представима в виде

$$a(x) = \sum_{i=1}^{2d+1} \sigma_i \left(\sum_{j=1}^d f_{ij}(x_j) \right),$$

где $x = [x_1, \dots, x_d]^T$ — вектор описания объекта, функции $\sigma_i(\cdot)$ и $f_{ij}(\cdot)$ являются непрерывными, а f_{ij} не зависят от выбора a .

Согласно формулировке теоремы, функция $a(x)$ определена только на единичном кубе, а, следовательно, все элементы выборки должны лежать в нем. Это не является проблемой: всегда можно отмасштабировать признаки так, чтобы на обучающей выборке каждый признак принимал значения из отрезка $[0, 1]$.

Также следует отметить, что в теореме не указан конкретный вид функции σ_i и f_{ij} . Проблема конструирования нейронной сети остается сложной задачей до сих пор.

10.1.4. Однослойная нейронная сеть как единственный нейрон

Однослойная нейронная сеть, или нейрон, определяется выражением:

$$a(x, w) = \sigma(w^T x) = \sigma\left(\sum_{j=1}^d w_j^{(1)} x_j + w_0^{(1)}\right),$$

где σ — функция активации, w — вектор параметров (весов), x — вектор описания объекта. Функция активации должна быть непрерывной, монотонной и, желательно, дифференцируемой функцией.

Следует обратить внимание, что для удобства $x \in \mathbb{R}^{d+1}$ дополнен постоянным на всех объектах признаком $x_0 = 1$. Соответствующий ему вклад в скалярное произведение $w^T x$ равен w_0 .

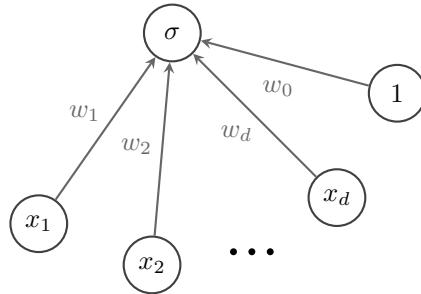


Рис. 10.3: Нейрон можно изобразить в виде вершины графа: он характеризуется своей функцией активации, имеет один выход и множество входов.

10.1.5. Функции активации: нейронная сеть как линейная модель

При решении задачи линейной регрессии в качестве функции активации можно выбрать тождественную функцию $\sigma = id$, тогда на выходе нейрона будет:

$$a(x, w) = w^T x,$$

то есть нейрон будет представлять собой линейный регрессор.

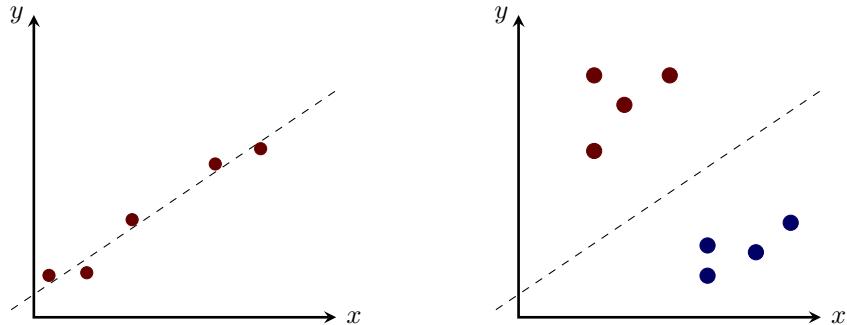


Рис. 10.4: Нейрон является линейным регрессором (график слева) или линейным классификатором (справа), если выбрать в качестве функции активации тождественную функцию или sign соответственно.

При решении задачи линейной классификации в качестве функции активации следует использовать пороговую функцию $\sigma = \text{sign}$. В таком случае нейрон:

$$a(x, w) = \text{sign}(w^T x)$$

является линейным классификатором.

10.1.6. Функции активации: модель логистической регрессии

Однослойная нейронная сеть является моделью логистической регрессии, если в качестве функции активации выбрана сигмоидная функция активации σ :

$$a(x, w) = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)}.$$

Эта функция определяет вероятность принадлежности некоторого заданного объекта x к классу $y = 1$ при заданных параметрах w :

$$\sigma(w^T x) = P(y = 1 | w, x).$$

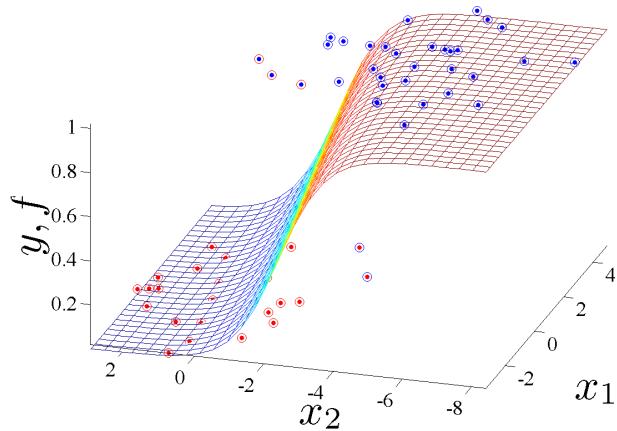


Рис. 10.5: Аппроксимирующая поверхность на графике построена с использованием сигмоидной функции активации.

Если количество классов равно K :

$$y = [y^1, \dots, y^K]^T,$$

то необходимо использовать сеть из K нейронов, каждый из которых вычисляет вероятность принадлежности к своему классу. В качестве функции активации используется softmax:

$$\sigma = \text{softmax}(w_1^T x, \dots, w_K^T x) = \frac{\exp(w_k^T x)}{\sum_{i=1}^K \exp(w_i^T x)}.$$

10.1.7. Функции активации: другие примеры

Гиперболический тангенс в качестве функции активации:

$$\tanh(x) = \frac{\exp(2x) - 1}{\exp(2x) + 1}$$

представляет собой дифференцируемую альтернативу пороговой функции $\text{sign}(\cdot)$.

Существует еще более мягкий вариант: функция активации

$$\text{softsign}(x) = \frac{x}{1 + |x|},$$

как и $\tanh(x)$ стремится к ± 1 , но в отличие от него, сходится к этим значениям не так быстро.

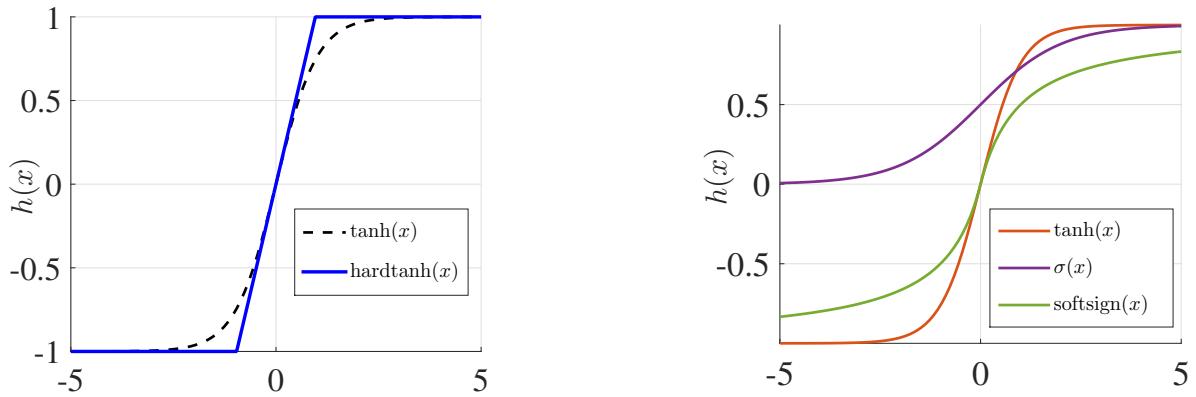


Рис. 10.6: Графики различных функций активации

В сетях глубокого обучения, в более сложных нейронных сетях, используются функция-выпрямитель Rectified linear unit (ReLU):

$$\text{ReLU}(f) = \max(0, f).$$

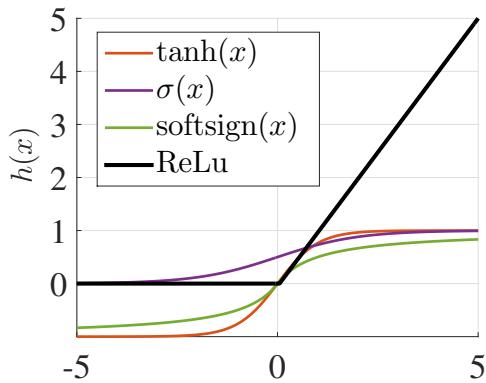


Рис. 10.7: Графики функций активации: $\text{ReLU}(f)$ и ее дифференцируемого приближения.

Эту функция ведет себя также, как работает диод в радиоэлектрических цепях. Данная функция не дифференцируема, но имеет следующее дифференцируемое приближение:

$$\ln(1 + \exp(f))$$

10.2. Многослойная нейронная сеть. Функция ошибки

В этом уроке будет рассказано про двухслойную нейронную сеть и про различные функции ошибок.

10.2.1. Пределы применимости однослойных сетей

Однослойные сети применимы только для линейно разделимых выборок.

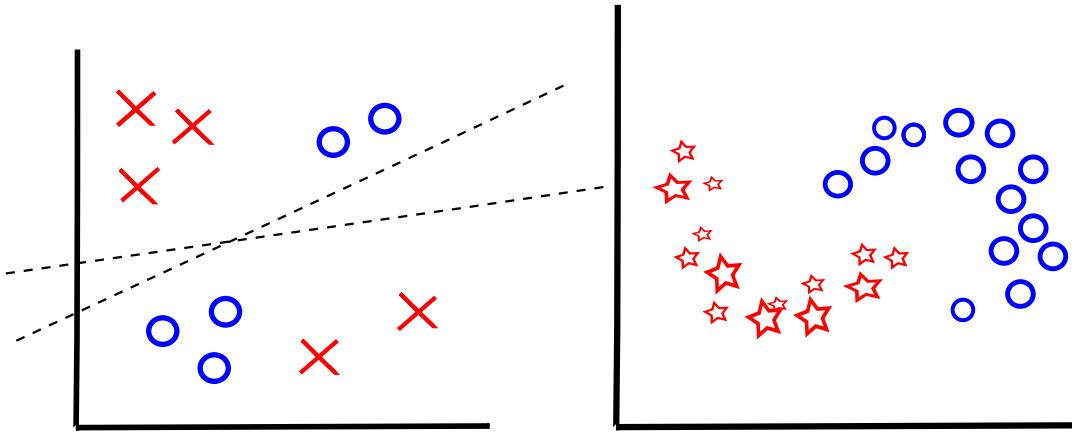


Рис. 10.8: Два примера линейно неразделимых выборок.

10.2.2. Двухслойная нейронная сеть

Двухслойная нейронная сеть представляет собой линейную комбинацию из D нейронов (однослойных нейронных сетей):

$$a(x, w) = \sigma^{(2)} \left(\sum_{i=1}^D w_i^{(2)} \cdot \sigma^{(1)} \left(\sum_{j=1}^d w_{ji}^{(1)} x_j + w_{0i}^{(1)} \right) + w_0^{(2)} \right).$$

Это выражение можно переписать в векторных обозначениях:

$$a(x, w) = \sigma^{(2)} \left(w^{(2)} \sigma^{(1)} \left([w_1^{(1)} x, \dots, w_D^{(1)} x] \right) \right).$$

Вектор w параметров нейронной сети получается соединением всех параметров нейронной сети на всех слоях:

$$w = \{W^{(1)}, w^{(2)}\},$$

где:

$$\begin{aligned} W^{(1)} &= [w_0^{(1)}, w_1^{(1)}, \dots, w_d^{(1)}]^T \in \mathbb{R}^{(d+1) \times D}, \\ w_i^{(1)} &= [w_{0i}^{(1)}, w_{1i}^{(1)}, \dots, w_{di}^{(1)}]^T \in \mathbb{R}^{d+1}, \quad w^{(2)} = [w_0^{(2)}, w_1^{(2)}, \dots, w_D^{(2)}]^T \in \mathbb{R}^{D+1}. \end{aligned}$$

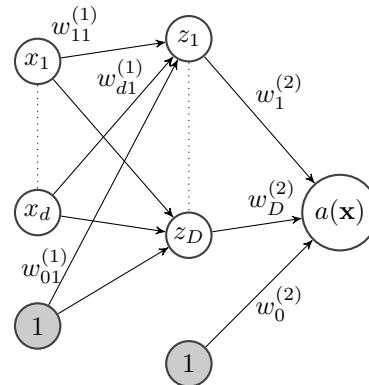


Рис. 10.9: Изображение двухслойной нейронной сети с помощью направленного графа.

Продолжая по аналогии, можно получить трехслойную и многослойную нейронные сети.

10.2.3. Разделяющая способность многослойной нейронной сети

Теорема (Универсальная теорема аппроксимации, K. Hornik, 1991) Для любой непрерывной функции $f(x)$ найдется нейронная сеть $a(x)$ с линейным выходом $a(x, W) = \sigma^{(M)}(x)$, где

$$f^{(k)}(x) = w_0^{(k)} + W^{(k)}z^{(k-1)}(x), \quad z^{(k)}(x) = \sigma^{(k)}(f^{(k-1)}(x)),$$

аппроксимирующая $f(x)$ с заданной точностью.

Теорема выполняется для различных функций активации, в частности для функции сигмоид и функции гиперболический тангенс. Чтобы получить требуемую аппроксимацию необходимо определить оптимальные значения параметров w^* .

10.2.4. Разделяющая поверхность: проблема переобучения

Качество аппроксимации целевой переменной y функцией $a(x, w)$ зависит от выбора параметров w .

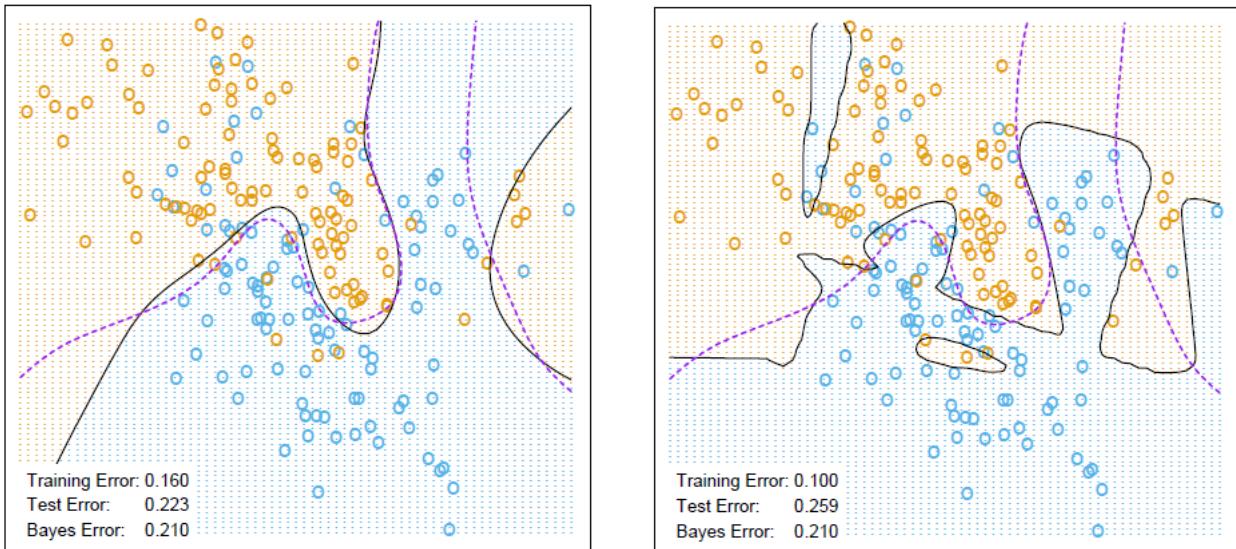


Рис. 10.10: Разделяющая поверхность нейронной сети в случае оптимальных и неоптимальных значений параметров.

В случае оптимальных значений параметров, как это видно по графику (левый), разделяющая поверхность отнесла несколько объектов не к тому классу, к которому они принадлежат. Тем не менее именно эту разделяющую поверхность следует считать оптимальной.

Действительно, в случае неоптимальных значений параметров, веса нейросети подстроены таким образом, что текущая выборка классифицируется идеально: каждый объект принадлежит области своего класса. Однако при изменении состава выборки разделение уже будет неверным. Такая нейросеть, которая корректно аппроксимирует обучающую выборку, но плохо справляется с контрольной, называется переобученной.

10.2.5. Функции ошибки нейронной сети

Пусть $Q(w)$ — функция ошибки, которая зависит как от конфигурации w нейронной сети, так и от ее состава. Задача нахождения оптимальных параметров w^* , то есть таких, при которых ошибка нейронной сети минимальна, имеет вид:

$$w^* = \operatorname{argmin}_w Q(w).$$

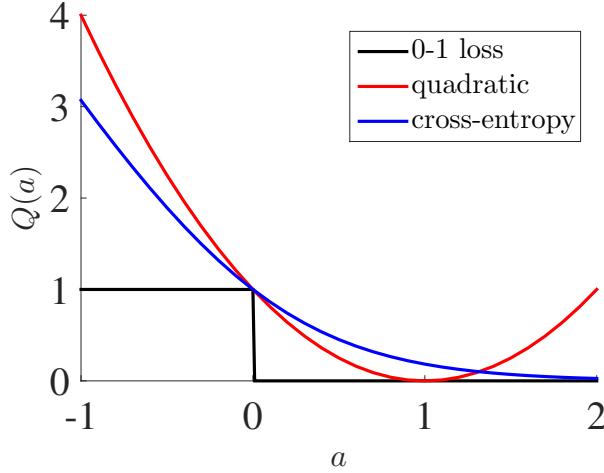


Рис. 10.11: Графики функций потерь

Выбор функции ошибки зависит от специфики решаемой задачи:

- В задаче регрессии (при решении различных прикладных задач) в качестве функции ошибки часто используется функция «галочка»:

$$Q(w) = \sum_{i=1}^{\ell} |a(x_i, w) - y_i|.$$

Но, к сожалению, эта функция не является дифференцируемой.

- Дифференцируемой функцией ошибки, используемой в задачах регрессии, является, например, квадратичная функция ошибки:

$$Q(w) = \sum_{i=1}^{\ell} (a(x_i, w) - y_i)^2.$$

- В задачах классификации используется функция ошибки «0-1 loss», которая равна числу несовпадений между восстановленными метками классов и фактическими:

$$Q(w) = \sum_{i=1}^{\ell} [\text{sign } a(x_i, w) \neq y_i].$$

- Дифференцируемая функция ошибки для задачи классификации — кросс-энтропия, функция наибольшего правдоподобия в задаче логистической регрессии:

$$Q(w) = - \sum_{i=1}^{\ell} (y_i \ln a(x_i, w) + (1 - y_i) \ln(1 - a(x_i, w))).$$

10.3. Оптимизация параметров нейронной сети

10.3.1. Задача оптимизации

Задача оптимизации для нахождения оптимальных значений параметров:

$$w^* = \operatorname{argmin}_w Q(w),$$

где Q — функция ошибки, которая зависит от выборки, структуры нейросети (числа слоёв, нейронов и видов функций активаций) и значения вектора параметров w .

На следующем графике изображено характерное поведение функции ошибки при изменении значений двух параметров.

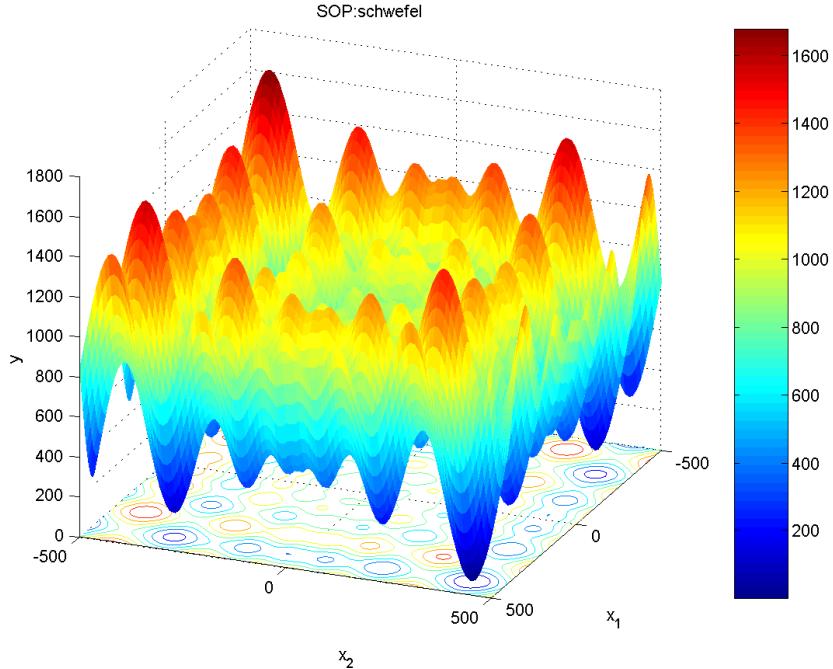


Рис. 10.12: Функция ошибки в зависимости от двух параметров.

Как отчетливо видно по графику, функция ошибки может иметь значительное число локальных минимумов. Задача является нетривиальной даже в пространстве малой размерности. При решении задачи оптимизации для нахождения оптимальных значений параметров есть два типа алгоритмов:

- Алгоритмы стохастической оптимизации:
 - Случайный перебор w_1, w_2, \dots ;
 - Генетический алгоритм оптимизации $w_1 \rightarrow w_2 \rightarrow \dots$;
 - Моделируемый отжиг, значения w задаются по расписанию;
- Алгоритмы градиентного спуска.

Применение алгоритмов градиентного спуска к данной задаче будет обсуждаться далее.

10.3.2. Локально-квадратичная аппроксимация функции ошибки

Алгоритм локальной аппроксимации функции ошибки минимизирует не саму функцию ошибки, а функцию, которая ее аппроксимирует в окрестности точки w^* . Например, удобно аппроксимировать исходную функцию квадратичной функцией:

$$Q(w) \approx Q(w^*) + \frac{1}{2}(w - w^*)^T \mathbf{H}(w - w^*).$$

На каждой итерации этого алгоритма ищется минимум приближенной функции и окрестность сдвигается в сторону более оптимальных значений.

10.3.3. Градиентный спуск

Алгоритм градиентного спуска предполагает, что функция ошибки является дифференцируемой (например квадратичная или кросс-энтропийная) и можно вычислить ее значение $Q(w)$ и значение ее градиента $\nabla_w Q(w)$.

При использовании градиентного спуска следует помнить, что он может найти как точку глобального минимума, так и точку локального минимума. Градиентный спуск — итеративная процедура нахождения параметров:

$$w_{k+1} = w_k - \alpha \sum_{i=1}^{\ell} \nabla_w \varphi(w_k, x_i),$$

где $\varphi(w_k, x_i)$ — ошибка на одном объекте, α — размер шага, k — номер итерации. Итеративный процесс останавливается, когда стабилизируется значение функции ошибки.

10.3.4. Стохастический градиентный спуск

Стохастический градиентный спуск, вариант метода градиентного спуска, заключается в том, что градиент считается не на всех элементах выборки, а только на одном случайно выбранном объекте x_i :

$$w_{k+1} = w_k - \alpha \nabla_w \varphi(w_k, x_k).$$

Итерации также продолжаются до стабилизации функции ошибки.

10.3.5. Обратное распространение ошибки (backprop)

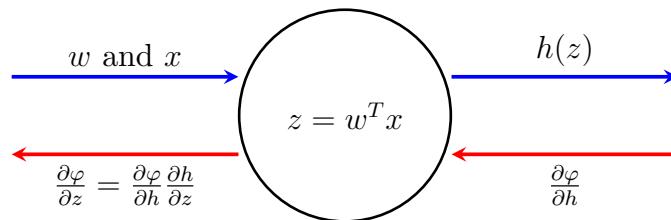
Очень популярный и быстрый алгоритм — это алгоритм обратного распространения ошибки backpropagation, или backprop. Для каждого нейрона сети $h(w^T x)$ и одного объекта выборки x вычисляется:

- Взвешенная сумму входных сигналов $z = w^T x$.
- Значение на выходе $h(z)$ (прямое распространение).
- Производная ошибки $\frac{\partial \varphi(z)}{\partial z}$ по значению нейрона и его параметрам. Эта производная зависит от значений последующих нейронов и называется обратным распространением.

По обратному распространению можно вычислить градиент функции ошибки по w :

$$\nabla_w \varphi(z) = \frac{\partial \varphi(z)}{\partial z} \nabla_w z = \frac{\partial \varphi(z)}{\partial z} x,$$

а затем подкорректировать веса аналогично тому, как это делалось в градиентных методах.



Существует так называемая проблема затухания градиента. При больших значениях входов нейронов $|x|$ значения производной функции активации $h'(x) \rightarrow 0$, например для сигмоидной функции $h(x) = \text{sigmoid}(x)$ или гиперболического тангенса $h(x) = \tanh(x)$. Следовательно, в данном случае градиент $\frac{\partial \varphi}{\partial z^{(s)}}$ не будет распространяться на предыдущие слои.

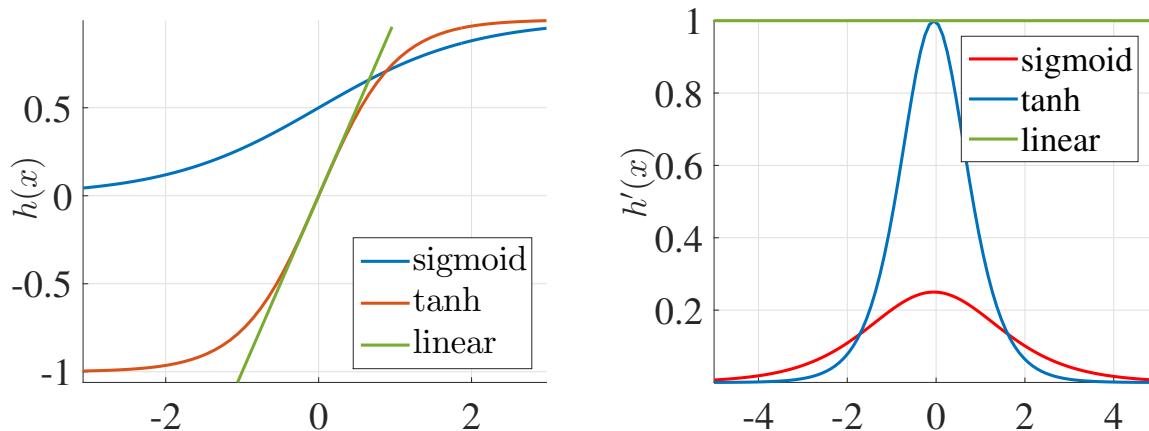


Рис. 10.13: Графики некоторых функций активаций и их производных.

Преимущества алгоритма обратного распространения ошибки:

- Градиент вычисляется за время, сравнимое с временем вычислением сети.
- Алгоритм подходит для многих дифференцируемых функций активации.
- Не обязательно использовать всю обучающую выборку.

Недостатки алгоритма обратного распространения ошибки:

- Возможна медленная сходимость к решению.
- Решение может быть локальным, а не глобальным минимумом.
- Возможно переобучение сети.

10.4. Регуляризация и прореживание нейронной сети

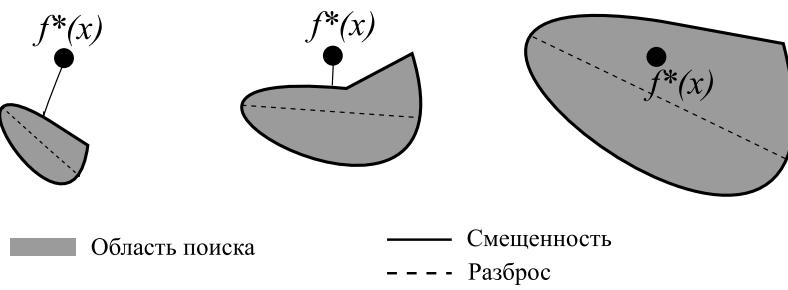
10.4.1. Регуляризация

Чтобы избежать переобучения, необходимо модифицировать задачу оптимизации: ввести штраф за большие значения весов:

$$w^* = \operatorname{argmin}_w \left[Q(w) + \tau \sum_{i,j,k} (w_{ij}^{(k)})^2 \right],$$

где τ — коэффициент регуляризации, который контролирует жесткость ограничений параметров w . При этом, чем меньше τ , тем точнее функция $a(x)$ описывает выборку.

На следующем рисунке изображены характерные смещенностъ, разброс и область поиска решения в пространстве параметров в случае различных значений параметра регуляризации τ .



На рисунке слева изображен случай, когда коэффициент регуляризации имеет самое большое значение. Область поиска решения в пространстве параметров мала, но велика смещенность параметра от минимального значения функции ошибки.

Справа изображена противоположная ситуация: коэффициент τ мал, область поиска оптимальных параметров велика и смещенности нет. Но, возможно, нейронная сеть окажется переобученной. В центре рисунка изображен компромиссный вариант.

На следующем графике изображена зависимость значений параметров нейронной сети от τ .

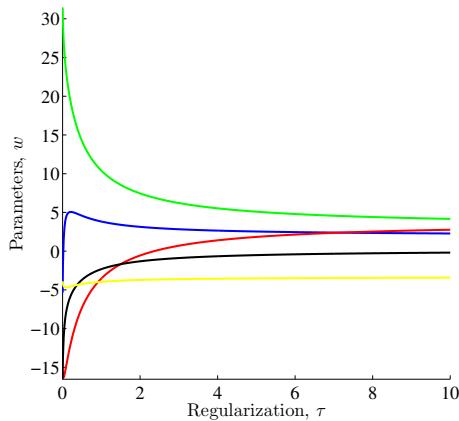


Рис. 10.14: Значения параметров сети в зависимости от τ .

По графику отчетливо видно, что регуляризация не снижает число параметров и не упрощает структуру сети. Более того, по мере увеличения τ параметры перестают изменяться.

10.4.2. Прореживание сети

Для снижения числа параметров предлагается исключить некоторые нейроны или связи. Принцип исключения следующий: если функция ошибки не изменяется, то нейронную сеть можно упрощать и дальше.

На следующем графике изображена характерная зависимость функции ошибки от числа удаленных параметров:

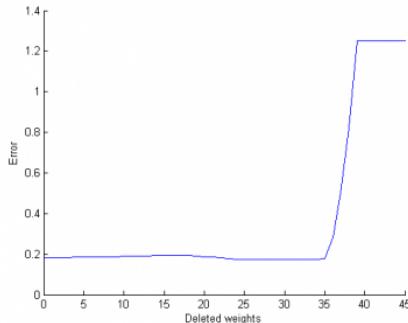


Рис. 10.15: Зависимость функции ошибки $Q(w)$ от числа удаленных связей.

По этому графику отчетливо видно, что функция ошибки начала изменяться, только когда значительное число связей было удалено. Связи (параметры) следует удалять исходя из следующих соображений. Параметр можно удалить, если:

- его значение близко к нулю.
- соответствующий сигнал обладает большой дисперсией, то есть реагирует на шум в данных.
- его удаление практически не меняет функцию ошибки.

Последний вариант следует рассмотреть подробнее: это так называемы «метод оптимального разрушения мозга» (optimal brain damage), или метод оптимального прореживания. Функцию ошибки можно разложить в ряд Тейлора в окрестности оптимума:

$$Q(w + \Delta w) = Q(w) + \mathbf{g}^T(w)\Delta w + \frac{1}{2}\Delta w^T \mathbf{H} \Delta w + o(\|\Delta w\|^3),$$

где \mathbf{H} матрица Гессе, а $\mathbf{g}^T(w) = 0$, поскольку разложение идет в окрестности оптимума.

В методе оптимального прореживания необходимо исключить один параметр w_j и, возможно, подкорректировать остальные веса w таким образом, чтобы приближенное выражение для изменения функции ошибки:

$$\Delta Q(w, \Delta w) \approx \frac{1}{2}\Delta w^T \mathbf{H} \Delta w$$

было минимальным. Исключению j -го параметра соответствует выполнение условия:

$$\Delta w_j + w_j = 0.$$

Таким образом, чтобы определить исключаемый признак, требуется решить задачу условной оптимизации:

$$j = \operatorname{argmin}_j \left(\min_{\Delta w} (\Delta w^T \mathbf{H} \Delta w \mid \Delta w_j + w_j = 0) \right).$$

Можно показать, что для этого параметра w_j будем минимальным значение функции выпуклости:

$$L_j = \frac{w_j^2}{2\mathbf{H}_{jj}^{-1}}.$$

10.4.3. Построение нейронной сети

Нейронная сеть может работать в двух режимах:

- **Режим обучения:** В этом режиме задается структура нейронной сети и оптимизируются ее параметры.
- **Режим эксплуатации:** вычисление значений $a(x, w^*)$ при фиксированных значениях параметров.

Для того чтобы задать нейронную сеть, требуется указать число слоев, число нейронов в каждом слое, тип функций активации в каждом слое, тип функции ошибки. Также желательно, чтобы подготовленная выборка не содержала пропусков, а признаки были отнормированы.

10.4.4. Стабилизация параметров сети

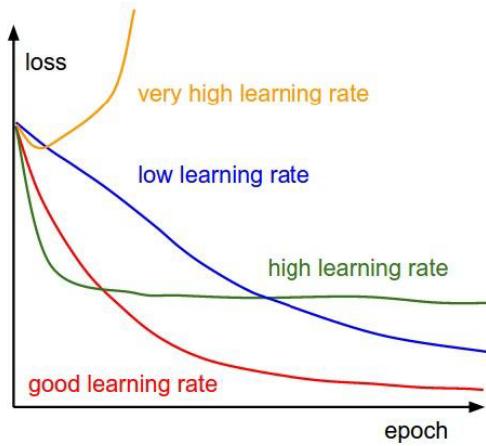


Рис. 10.16: Зависимость ошибки от номера итерации.

По скорости обучения можно судить о соответствии выборки и нейронной сети:

- Если нейронная сеть будет слишком сложна, то она быстро переобучится, как, например, показывает желтая линия.
- Если выборка будет сложна или сильно зашумлена для нейронной сети, то нейронная сеть будет обучаться медленно, как показывает синяя линия.
- Если выборка по сложности соответствует нейронной сети, то, скорее всего, мы увидим красную линию — хорошую скорость обучения.
- Важно также следить за разницей между значениями функции ошибки на обучении и на контроле. Эта разница не должна быть существенной. Если она велика, то это значит, что нейронная сеть переобучилась и следует изменить ее сложность.

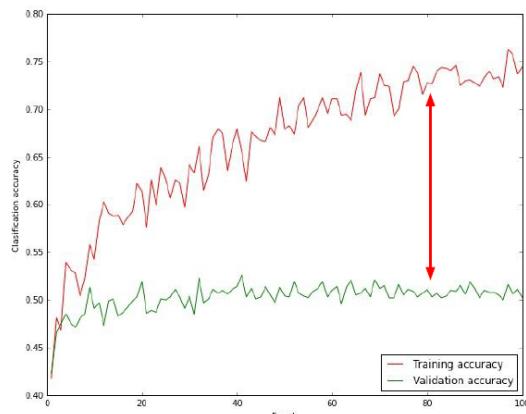


Рис. 10.17: Существенная разница между значениями ошибки на обучении и на контроле говорит о переобученности нейронной сети.

Урок 11

Байесовская классификация и регрессия

11.1. Спам-фильтр на основе байесовского классификатора

Этот раздел посвящен наивному байесовскому классификатору на примере практической задачи фильтрации спама.

11.1.1. Задача фильтрации спама

Фильтр спама представляет собой бинарный классификатор:

$$Y = \begin{cases} \text{spam}, & \text{спам} \\ \text{ham}, & \text{не спам} \end{cases}.$$

Первые спам-фильтры являлись наивными байесовскими классификаторами.

Примеры спамных писем:

- Hi! :) Purchase Exclusive Tabs Online <http://...>
- We Offer Loan At A Very Low Rate Of 3%. If Interested, Kindly Contact Us, Reply by email ...@hotmail.com
- Купите специализацию Машинное обучение и анализ данных от МФТИ и Яндекса с супер-скидкой 0.99%! Станьте Data Scientist за 5 месяцев!

Отчетливо видно, что некоторые слова особенно часто встречаются в спаме.

Пусть есть коллекция писем, среди которых n_s писем — спам, а n_h — не спам. Возникает идея подсчитать для каждого такого слова w количество n_{ws} писем со спамом и количество n_{wh} писем без спама, в которых есть это слова. Тогда можно оценить вероятность появления каждого слова в спамном и неспамном письме:

$$P(w|\text{spam}) = \frac{n_{ws}}{n_s}, \quad P(w|\text{ham}) = \frac{n_{wh}}{n_h}.$$

Пусть теперь требуется выяснить является ли некоторый новый текст, содержащий ключевые слова w_1, \dots, w_N , спамом или нет. Можно оценить следующие вероятности порождения текста, если известно, что он принадлежит какому-либо из классов:

$$\begin{aligned} P(\text{text}|\text{spam}) &= P(w_1|\text{spam})P(w_2|\text{spam})\dots P(w_N|\text{spam}), \\ P(\text{text}|\text{ham}) &= P(w_1|\text{ham})P(w_2|\text{ham})\dots P(w_N|\text{ham}). \end{aligned}$$

Такую оценку можно сделать только в случае, когда вероятность вхождения разных слов в текст — независимые события. Это достаточно наивное предположение, поэтому классификатор называется «наивный байесовский классификатор».

11.1.2. Идея наивного байесовского классификатора

В качестве алгоритма можно использовать следующий: выбрать такой класс, вероятность порождения текста в котором больше:

$$a(\text{text}) = \operatorname{argmax}_y P(\text{text}|y).$$

Это почти правильный алгоритм, но, поскольку текст уже известен, более правильно оценивать вероятность $P(y|text)$ того, что этот текст принадлежит какому-то из классов:

$$a(text) = \operatorname{argmax}_y P(y|text).$$

Эту вероятность можно вычислить используя теорему Байеса:

$$P(y|text) = P(text|y)P(y)/P(text)$$

Тогда, поскольку $P(text)$ не содержит зависимость от y :

$$a(text) = \operatorname{argmax}_y P(text|y)P(y)/P(text) = \operatorname{argmax}_y P(text|y)P(y).$$

11.1.3. Спам фильтр на наивном байесовском классификаторе

Для того, чтобы построить спам фильтр на наивном байесовском классификаторе, на стадии обучения необходимо оценить вероятности вхождения слов в тексты каждого из классов:

$$P(w|spam) = \frac{n_{ws}}{n_s}, \quad P(w|ham) = \frac{n_{wh}}{n_h}.$$

На стадии применения классификатора к текстам необходимо выбрать такой класс $y \in \{spam, ham\}$, для которого произведение:

$$P(y)P(text|y) = P(y)P(w_1|y)...P(w_N|y)$$

максимально. Именно к этому классу и следует отнести текст.

11.1.4. Что не было учтено?

При построении такого спам-фильтра не были учтены следующие моменты:

- Никак не использована информация, содержащаяся в заголовке письма и адресе отправителя.
- Если слово w не встречается ни в одном из обучающих текстов для какого-то класса, его вероятности $P(w|y)$ сразу оценивается нулем. А значит вероятность того, что текст принадлежит классу y сразу оценивается нулем, что может быть весьма поспешным решением.
- Допустим есть слово w_1 , которое не входило в обучающие тексты первого класса (со спамом), и слово w_2 , которое не входило в обучающие тексты второго класса (без спама). Тогда, если оба этих слова содержатся в некотором тексте, обе вероятности $P(y_1|spam)$ и $P(y_2|ham)$ будут равны нулю, а значит нельзя будет отнести текст к какому-либо из классов.

11.2. Наивный байесовский классификатор

11.2.1. Байесовский классификатор

Пусть некоторый объект имеет вектор признаков x . Необходимо определить, к какому классу y относится этот объект. Байесовский классификатор $a(x)$ относит объект к такому классу, вероятность которого при условии, что реализовался данный объект, максимальна:

$$a(x) = \operatorname{argmax}_y P(y|x) = \operatorname{argmax}_y P(x|y)P(y).$$

Здесь была использована теорема Байеса: $P(y|x) = P(x|y)P(y)/P(x)$.

11.2.2. Необходимость использования теоремы Байеса

Непосредственное вычисление $P(y|x)$ заключается в том, что необходимо рассмотреть множество объектов, которые имеют признаковое описание x , и найти долю класса y среди этого множества. Но возможных признаковых описаний огромное количество, а значит вряд ли в обучающей выборке будет достаточное количество объектов для всякого возможного x . Таким образом, не получится вычислять $P(y|x)$ непосредственно и приходится применять теорему Байеса.

Теорема Байеса позволяет переходить к $P(x|y)$, то есть фактически к плотности распределения x при условии класса y (в случае вещественных признаков). Последнюю величину уже можно оценивать по обучающей выборке.

Применение классификатора происходит следующим образом:

$$a(x) = \operatorname{argmax}_y P(x|y)P(y).$$

11.2.3. Проблема нехватки данных

Однако все еще стоит проблема нехватки данных. Пусть, например, обучающая выборка состоит из 100 000 объектов, а пространство признаков имеет размерность 10 000. В этом случае восстановить плотность как функцию от всех признаков достаточно затруднительно.

11.3. Восстановление распределений (часть 1)

Непосредственно восстановить распределение $P(x|y)$ не получается из-за рассмотренной выше проблемы нехватки данных.

11.3.1. Наивный байесовский классификатор

Одно из решений проблемы нехватки данных — использование «наивного» байесовского классификатора, то есть байесовского классификатора:

$$a(x) = \operatorname{argmax}_y P(y|x) = \operatorname{argmax}_y P(x|y)P(y).$$

и «наивной» гипотезы, что плотность распределения расписывается в произведение плотностей по каждому признаку:

$$P(x|y) = P(x_{(1)}|y)P(x_{(2)}|y)\dots P(x_{(N)}|y),$$

где $x_{(k)}$ — k -ый признак объекта x .

Эта гипотеза выполняется только в случае, если признаки независимые. Это далеко не всегда так, но с некоторой степенью точности таким приближением пользоваться можно.

11.3.2. Восстановление распределений $P(x_{(k)}|y)$

Таким образом, при обучении необходимо определить по обучающей выборке распределения $P(x_{(k)}|y)$ и априорные вероятности классов $P(y)$.

Оценить априорные вероятности классов на основе выборки можно следующим образом:

$$P(y) \approx \frac{\ell_y}{\ell},$$

где ℓ_y — количество объектов класса y в обучающей выборке, а ℓ — размер обучающей выборки. Если соотношение долей классов в обучающей выборке не отражает их реальное соотношение, априорные вероятности классов должны быть взяты из внешних данных.

Распределение $P(x_{(k)}|y)$ можно оценить как долю объектов с данным значением признака $x_{(k)}$ среди объектов класса y :

$$P(x_{(k)}|y) = \frac{1}{\ell_y} \#(x_{(k)}, y).$$

Таким образом, для бинарных признаков:

$$P(x_{(k)} = 0|y) = \frac{1}{\ell_y} \#(x_{(k)} = 0, y), \quad P(x_{(k)} = 1|y) = \frac{1}{\ell_y} \#(x_{(k)} = 1, y).$$

11.3.3. Пример: классификация текстов

Классификатор текстов можно построить следующим образом. По обучающей выборке строится словарь всех входящих в тексты обучающей выборки слов. Каждый текст будет характеризоваться вектором из бинарных признаков: $x_{(k)} = 1$, если слово w_k присутствует в тексте, а если не присутствует — $x_{(k)} = 0$.

После этого можно восстановить распределения как это описано выше:

$$P(x_{(k)} = 0|y) = \frac{1}{\ell_y} \#(x_{(k)} = 0, y), \quad P(x_{(k)} = 1|y) = \frac{1}{\ell_y} \#(x_{(k)} = 1, y).$$

После этого можно применить наивный байесовский классификатор и таким образом решить задачу классификации.

Следует обратить внимание, что получается не совсем то, что было в примере со спамом. Предлагается самостоятельно подумать, почему это так.

11.3.4. Сглаживание вероятностей

Если в обучающей выборке среди множества объектов определенного класса y никогда не встречалось какое-то значение t некоторого признака $x_{(k)}$, то вероятность $P(x_{(k)} = t|y) = 0$. Поскольку в выражении, которое требуется максимизировать, стоит произведение таких вероятностей, все это выражение будет равно нулю. Таким образом, любой объект, только на основании того, что значение признака $x_{(k)} = t$, не будет отнесен к классу y , что, вообще говоря, неправильно.

Избежать такой ситуации можно с помощью сглаживания вероятности, например следующим образом:

$$P(x_{(k)} = 1|y) = \frac{\#(x_{(k)} = 1, y) + a}{\ell_y + a + b}, \quad P(x_{(k)} = 0|y) = \frac{\#(x_{(k)} = 0, y) + b}{\ell_y + a + b}.$$

Константы a и b выбираются таким образом, что качество алгоритма получалось наибольшим.

11.4. Восстановление распределений (часть 2)

Рассмотренный ранее способ восстановления распределений для бинарных признаков не годится в случае вещественных признаков.

11.4.1. Параметрическое восстановление распределения

Однако можно предположить, что распределение имеет какой-то определенный вид: пуассоновское, экспоненциальное или нормальное, и попробовать восстановить его. Это метод называется методом параметрического восстановления распределений.

Пусть, например, рассматривается нормальное распределение:

$$X \sim \mathcal{N}(\mu, \sigma^2).$$

Плотность распределения имеет вид

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

полностью определяется значениями двух параметров: математического ожидания μ и дисперсии σ^2 . С помощью оценок максимального правдоподобия для этих параметров:

$$\hat{\mu} = \frac{1}{N} \sum_{i=1}^N x_i, \quad \hat{\sigma}^2 = \frac{1}{N} \sum_{i=1}^N (x_i - \hat{\mu})^2$$

можно оценить параметры по обучающей выборке. Несмешанный вариант оценки для дисперсии:

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

Другой пример — распределение Бернулли. Это распределение характеризуется одним параметром — вероятность того, что случайная величина принимает значение 1. Этот параметр можно оценить как долю случаев, в которых случайная величина равнялась 1:

$$\hat{p} = \frac{1}{N} \sum_{i=1}^N [x_i = 1].$$

Также следует отметить, что рассмотренные ранее оценки распределения бинарных признаков — частный случай параметрического восстановления плотности.

11.4.2. Рекомендации по выбору распределений

Верны следующие общие рекомендации по выбору распределения при использовании метода параметрического восстановления:

- Если решается задача, связанная с текстами или какими-то другими разреженными дискретными признаками, то хорошо подходит мультиномиальное распределение.
- Если в задаче есть непрерывные признаки с небольшим разбросом, то можно попробовать использовать нормальное распределение.
- Для непрерывных признаков с большим разбросом нужны более «размазанные», нежели нормальное, распределения.

При этом не обязательно ограничиваться наивным байесовским классификатором. Проблему нехватки данных можно решать с помощью параметрической оценки многомерных распределений, но решение искать в каком-то узком классе так, чтобы оно определялось небольшим числом параметров.

Например, можно предположить, что распределение является многомерным нормальным распределением

$$p(x) = \frac{1}{(2\pi)^{\frac{n}{2}} \sqrt{\det \Sigma}} e^{-\frac{1}{2}(x-\mu)^T \Sigma^{-1}(x-\mu)}$$

и по выборке оценивать его параметры: вектор средних μ и матрицу ковариаций Σ . Причем количество вещественных параметров будет гораздо больше, чем в «наивном подходе». Действительно, в «наивном» подходе параметры — это n средних и n дисперсий, а в случае многомерного нормального распределения — вектор размера n и матрица размера $n \times n$. Поэтому из-за нехватки данных оценка каких-то параметров может получиться неверной, а также часто возникают неустойчивые операции, например обращение почти вырожденных матриц и так далее.

Существует также непараметрическое восстановление плотности, о котором будет подробнее рассказано в следующих курсах.

11.5. Минимизация риска

В этом разделе предложен другой взгляд на байесовскую классификацию, а также будет произведено обобщение на случай задачи регрессии.

11.5.1. Байесовская регрессия

Байесовский классификатор определяется выражением:

$$a(x) = \operatorname{argmax}_y P(x|y)P(y),$$

где x — признаковое описание, y — класс.

Применить такую же формулу в случае регрессии (в этом случае y — прогнозируемая величина) не получится, так как вряд ли получится восстановить распределение $P(x|y)$, поскольку y — вещественное число.

Если воспользоваться при решении задачи регрессии выражением:

$$a(x) = \operatorname{argmax}_y P(y|x),$$

то это будет соответствовать поиску максимума функции плотности по y при выбранном x . Не очевидно, что это будет хорошим решением задачи регрессии.

11.5.2. Штрафы за ошибки

Часто бывает необходимо по-разному штрафовать алгоритм за разные типы ошибок. Например, в задаче классификации нефтяных месторождений с двумя классами «есть нефть» и «нет нефти» ошибочный положительный результат — более критичная ошибка, так как бурение скважины требует огромных денежных и временных затрат.

В задачах регрессии штрафы за ошибки еще более естественны: так как искомую зависимость идеально восстановить невозможно, требуется именно минимально отклониться от нее. В задачах регрессии в качестве меры отклонения часто используются квадратичные потери (MSE) и сумма модулей отклонения (MAE):

$$MSE = \frac{1}{\ell} \sum_{i=1}^{\ell} (y_i - a(x_i))^2, \quad MAE = \frac{1}{\ell} \sum_{i=1}^{\ell} |y_i - a(x_i)|.$$

11.5.3. Более общий подход

Пусть для некоторого объекта x необходимо сделать прогноз $a(x)$. Какая именно задача, задача регрессии или классификации, рассматривается, не имеет значения. Пусть также y — правильный ответ, а функция $L(y, a(x))$ определяет величину ошибки алгоритма и задается в зависимости от рассматриваемой задачи и желаемых свойств алгоритма.

В задаче классификации можно использовать в качестве функции $L(y, a(x))$ индикатор того, что точный ответ y не совпадает с прогнозом $a(x)$:

$$L(y, a(x)) = [y \neq a(x)].$$

Такой выбор функции приведет к тому, что полученный классификатор будет уже рассмотренным ранее байесовским классификатором, но об этом будет рассказано позднее.

В задаче регрессии используется квадратичная функция:

$$L(y, a(x)) = (y - a(x))^2.$$

11.5.4. Оптимальный байесовский классификатор

Так называемый функционал риска $R(a, x)$ определяется как условное математическое ожидание потерь при известном x и ответе алгоритма a :

$$R(a, x) = \mathbb{E}(L(y, a(x))|x).$$

Можно строить ответы алгоритма таким образом, чтобы минимизировать ожидаемые потери:

$$a(x) = \operatorname{argmin}_s R(s, x).$$

В случае задачи классификации можно записать следующее:

$$R(s, x) = \mathbb{E}(L(y, s)|x) = \sum_{y \in Y} L(y, s) P(y|x)$$

$$a(x) = \operatorname{argmin}_s R(s, x) = \operatorname{argmin}_s \sum_{y \in Y} L(y, s) P(y|x) = \operatorname{argmin}_s \sum_{y \in Y} L(y, s) P(y) P(x|y).$$

Такой классификатор называется оптимальным байесовским классификатором, потому что он минимизирует ожидаемые потери. Реальный классификатор, конечно, не будет оптимальным из-за использования вероятностных оценок, а не истинных вероятностей.

11.5.5. Оптимальный байесовский регрессор

Для задачи регрессии выражения выглядят аналогичным образом:

$$R(s, x) = \mathbb{E}(L(y, s)|x) = \int_{y \in Y} L(y, s) P(y|x) dy$$

$$a(x) = \operatorname{argmin}_s R(s, x) = \operatorname{argmin}_s \int_{y \in Y} L(y, s) P(y|x) dy.$$

На практике данный результат используется не для решения задачи регрессии, а чтобы проанализировать разные функции потерь, о чём будет рассказано позднее.

11.5.6. Функционал среднего риска

Функционал среднего риска

$$R(a) = \mathbb{E}_x R(a(x), x)$$

позволяет оценить, насколько хорошо работает алгоритм в среднем, а не для конкретного x .

Для определенности дальше рассматривается случай задачи классификации объектов с дискретными признаками. Остальные случаи, например случаи задачи регрессии или задачи классификации объектов с непрерывными признаками, полностью аналогичны с точностью до замены суммы на интеграл.

В данной ситуации функционал среднего риска просто представляется взвешенной суммой возможных значений функционала риска, где в качестве весов выступают вероятности $P(x)$:

$$R(a) = \sum_{x \in X} R(a(x), x) P(x).$$

Поскольку $R(s, x) \geq \min_s R(s, x)$, верна следующая оценка снизу для $R(a)$:

$$R(a) = \sum_{x \in X} R(a(x), x) P(x) \geq \sum_{x \in X} \min_s R(s, x) P(x).$$

Эта нижняя оценка достигается, если $R(s, x) = \min_s R(s, x)$, то есть в уже знакомом случае оптимального байесовского классификатора. Таким образом, оптимальный байесовский классификатор минимизирует не только функционал риска, но и функционал среднего риска.

11.6. Минимизация риска и анализ функции потерь

Итак, уже в прошлом видео было анонсировано, что с помощью нового, более общего взгляда на байесовскую классификацию можно получить несколько интересных результатов.

11.6.1. Оптимальный байесовский классификатор

Можно показать, что оптимальный байесовский классификатор:

$$a(x) = \operatorname{argmin}_s R(s, x) = \operatorname{argmin}_s \sum_{y \in Y} L(y, s) P(y|x),$$

в случае, если функция потерь равна индикатору того, что прогноз алгоритма $a(x)$ не совпал с правильным ответом y :

$$L(y, a(x)) = [y \neq a(x)],$$

переходит в знакомый с начала урока байесовский классификатор.

Действительно, если подставить функцию потерь в минимизируемое выражение:

$$\sum_{y \in Y} L(s, y) P(y|x) = \sum_{y \in Y \setminus \{s\}} P(y|x) = \sum_{y \in Y} P(y|x) - P(s|x) \rightarrow \min_s \quad \Rightarrow \quad P(s|x) \rightarrow \max_s,$$

Таким образом, классификатор имеет вид:

$$a(x) = \operatorname{argmax}_y P(y|x) = \operatorname{argmax}_y P(y)P(x|y),$$

то есть является байесовским классификатором.

11.6.2. Квадратичная функция потерь в регрессии

Оказывается, такой подход годится для анализа различных функций потерь в задаче регрессии. Например, в случае квадратичной функции потерь:

$$\int_Y (t - y)^2 p(y|x) dy \rightarrow \min_t .$$

Минимум можно найти, если приравнять производную по t к нулю:

$$\frac{\partial}{\partial t} \int_Y (t - y)^2 p(y|x) dy = 2 \int_Y (t - y) p(y|x) dy = 2 \left(t \int_Y p(y|x) dy - \int_Y y p(y|x) dy \right) = 0.$$

Так как $p(y|x)$ — плотность вероятности, $\int_Y p(y|x) dy = 1$:

$$a(x) = t = \int_Y y p(y|x) dy = \mathbb{E}(y|x).$$

Таким образом, прогноз алгоритма должен равняться условному математическому ожиданию $\mathbb{E}(y|x)$.

11.6.3. Абсолютное отклонение

В случае, когда функция потерь — абсолютное отклонение:

$$\int_Y |t - y| p(y|x) dy \rightarrow \min_t,$$

выкладки производятся точно также. Единственный нюанс заключается в том, что модуль не имеет производной в нуле, поэтому точку $y = t$ следует заблаговременно исключить из области интегрирования (важно, что это не изменит значение интеграла):

$$\begin{aligned} \frac{\partial}{\partial t} \int_Y |t - y| p(y|x) dy &= \frac{\partial}{\partial t} \int_{Y \setminus \{t\}} |t - y| p(y|x) dy = \int_{Y \setminus \{t\}} \text{sign}(t - y) p(y|x) dy = \\ &= \int_{\{t > y\}} p(y|x) dy - \int_{\{t < y\}} p(y|x) dy = P(\{t > y\}|x) - P(\{t < y\}|x) = 0. \end{aligned}$$

Таким образом, учитывая, что $P(\{t = y\}|x) = 0$, можно получить:

$$P(\{t > y\}|x) = P(\{t < y\}|x) = \frac{1}{2}.$$

Другими словами, ответ алгоритма оценивает 1/2 квантиль (медиану).

11.6.4. Оценка вероятности

Рассматривается задача бинарной классификации $Y = \{0, 1\}$. Необходимо, чтобы алгоритм классификации оценивал вероятность того, что объект принадлежит к первому классу $p = P(1|x)$. Оказывается, что получить требуемый результат можно, выбрав в качестве функции потерь так называемую функцию Log loss:

$$L(y, a(x)) = -y \ln a(x) - (1 - y) \ln(1 - a(x))$$

Условие минимальности потерь тогда принимает вид:

$$\sum_{y \in Y} \left(-y \ln t - (1 - y) \ln(1 - t) \right) P(y|x) = -(1 - p) \ln(1 - t) - p \ln t \rightarrow \min_t,$$

где использовано обозначение $p = P(1|x)$. Минимум можно найти вычислением производной по t :

$$\frac{\partial}{\partial t} \left(-(1 - p) \ln(1 - t) - p \ln t \right) = \frac{1 - p}{1 - t} - \frac{p}{t} = \frac{t - p}{(1 - t)t} = 0.$$

Получается требуемый результат:

$$a(x) = t = p,$$

то есть ответ алгоритма t должен равняться вероятности p того, что объект принадлежит к первому классу.

11.6.5. Обоснование метода анализа функции потерь

Следует напомнить, что в байесовской классификации минимизируется именно функционал среднего риска:

$$R(a) = \mathbb{E}_{x,y} L(y, a(x)).$$

Поскольку ошибка Q на обучающей выборке является эмпирической оценкой функционала среднего риска:

$$Q = \frac{1}{\ell} \sum_{i=1}^{\ell} L(y_i, a(x_i)) \sim \mathbb{E}_{x,y} L(y, a(x)),$$

результаты приведенного выше метода анализа функции потерь остаются верны не только в случае использования байесовского классификатора или байесовской регрессии, но и для произвольного метода решения, в ходе которого минимизируется ошибка на обучающей выборке.

Урок 12

Метрические алгоритмы и SVM

12.1. Метод ближайших соседей

В этом разделе будут рассмотрены метрические алгоритмы, которые подразумевают, что в пространстве признаков было введено понятие расстояния, другими словами, определена метрика.

12.1.1. Метод ближайших соседей (kNN)

Самый простой метрический метод в задаче классификации — метод ближайшего соседа, суть которого заключается в том, что новая точка относится к такому же классу, что и ближайшая к ней точка из обучающей выборки.

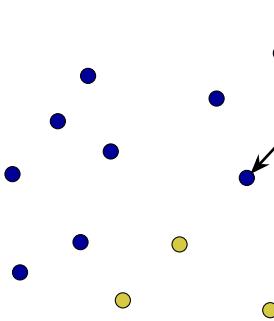


Рис. 12.1: Метод ближайшего соседа

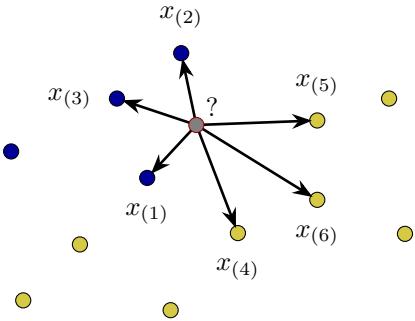


Рис. 12.2: Метод ближайших соседей ($k = 6$)

Для повышения надежности можно принимать решение по более чем одной точке. В этом и состоит метод ближайших соседей: объект относится к тому классу, к которому принадлежит большинство из его k ближайших соседей. Еще больше повысить надежность можно правильным образом определив веса в методе ближайших соседей. Веса могут зависеть как от номера соседа $w(x_{(i)}) = w(i)$, так и от расстояния до него $w(x_{(i)}) = w(d(x, x_{(i)}))$.

Во взвешенном kNN объект x относится к тому классу, взвешенная сумма по объектам из множества k ближайших соседей для которого больше:

$$a(x) = \operatorname{argmax}_{y \in Y} \sum_{i=1}^k [x_{(i)} = y] w(x_{(i)}).$$

12.1.2. Центроидный классификатор

Еще один простой метрический классификатор — центроидный классификатор. Сначала по обучающей выборке $\{(x_i, y_i)\}_{i=1}^m$ определяются «центры» всех классов (ℓ_y — количество объектов класса y):

$$\mu_y = \frac{1}{\ell_y} \sum_{i:y_i=y} x_i.$$

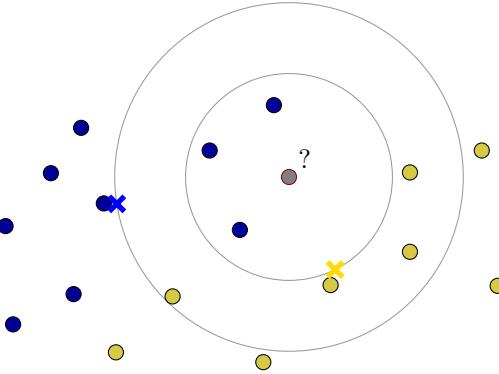


Рис. 12.3: Центроидный классификатор. Крестиками изображены центры классов.

После этого центроидный классификатор относит каждый новый объект x к тому классу, центр которого находится ближе всего в пространстве признаков к признаковому описанию нового объекта:

$$a(x) = \operatorname{argmin}_{y \in Y} d(\mu_y, x).$$

12.1.3. Взвешенный kNN для регрессии

Метод k ближайших соседей можно использовать для решения задачи регрессии. Пусть x — новый объект, который требуется классифицировать, а $x_{(i)}$ — i -ый ближайший сосед из обучающей выборки. Взвешенный kNN для задачи регрессии в таком случае определяется выражением:

$$a(x) = \frac{\sum_{i=1}^k w(x_{(i)})x_{(i)}}{\sum_{i=1}^k w(x_{(i)})}.$$

Следует обратить внимание на выбор весовой функции, поскольку это сильно влияет на качество решения.

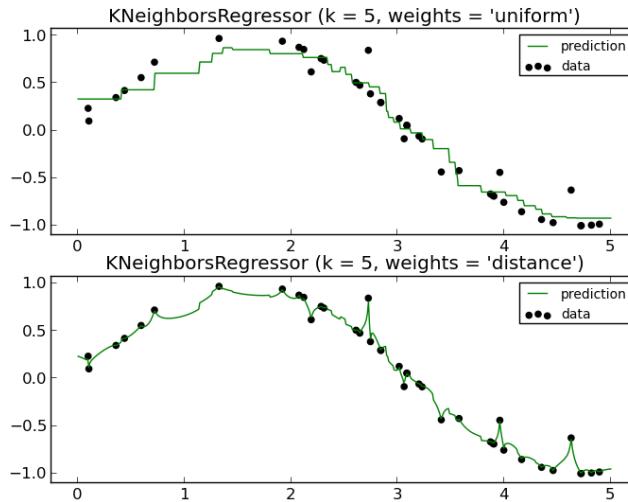


Рис. 12.4: Метод kNN в задаче регрессии в случае различных весовых функций.

Например, если в качестве веса в задаче регрессии использовать величину, обратную к расстоянию, то в результате получится переобученная модель. Это связано с тем, что, если объект уже есть в обучающей выборке, то он будет входить в сумму практически с бесконечным весом и алгоритм вернет для него то же значение, что и было в обучающей выборке.

12.2. Настройка параметров в kNN

Данный раздел посвящен вопросу подбора параметров в методе ближайших соседей так, чтобы качество работы алгоритма было наилучшим. К числу параметров относятся: количество соседей, весовые функции, метрика и так далее.

Проверять качество работы алгоритма с выбранными параметрами лучше не на обучающей выборке, а на отложенной. Также можно использовать кросс-валидацию.

12.2.1. Количество соседей

Основной параметр в kNN — количество соседей.

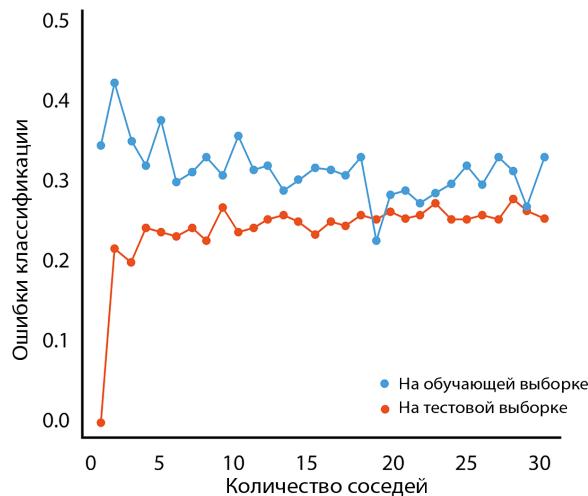


Рис. 12.5: Зависимость ошибки классификации от количества соседей в kNN для объектов из тестовой и обучающей выборок.

Если оценивать качество работы алгоритма по объектам из обучающей выборки, то, как это показывает график, оптимальное значение k будет равно 1. При этом значении k алгоритм совсем не ошибается на объектах обучающей выборки. Действительно, при классификации объекта из обучающей выборки, ближайшим к нему объектом из обучающей выборки будет он сам. Именно поэтому качество алгоритма нужно всегда проверять на тестовой выборке.

В общем зависимость от k следующая: сначала качество на контроле становится все лучше с ростом k , а затем качество начинает ухудшаться.



Рис. 12.6: Ошибка классификации для объектов из тестовой выборки.

Выбирать следует такое значение k , при котором достигается наилучшая оценка качества работы алгоритма на контроле.

12.2.2. Веса соседей как функция от номера

Если используется взвешенный kNN, то необходимо задать весовую функцию. Такая функция не должна возрастать с ростом номера объекта. Простейший вариант — это $w(x) = 1$. При выборе весовой функции всегда следует сначала пробовать его перед тем, как рассматривать более сложные варианты.

Если выбор $w(x) = 1$ не дает желаемых результатов, можно попробовать определить веса как функцию от номера соседа:

- $w(i) = q^i, \quad 0 < q < 1$
- $w(i) = \frac{1}{i}, \quad w(i) = \frac{1}{i+a}, \quad w(i) = \frac{1}{(i+a)^b}$
- $w(i) = 1 - \frac{i-1}{k}$ (не очень удачный вариант).

Функция, которая линейно зависит от номера соседа, не является хорошим выбором. Например, в случае $k = 4$, если 1 и 4 соседи некоторого объекта x принадлежат к первому классу, а 2 и 3 — ко второму, алгоритм не сможет классифицировать этот объект. Это, конечно, не значит, что эта функция вовсе не применима на практике, но эту её особенность следует учитывать.

12.2.3. Веса объектов как функция от расстояния

Другой способ определить весовую функцию — задать ее как функцию от расстояния. Ранее уже было сказано, что в задаче регрессии выбор весовой функции $w(d) = \frac{1}{d}$ приводит к переобученности. Грубо говоря, это было связано с тем, что при $d = 0$ значение весовой функции было бесконечно большим. Поэтому необходимо более аккуратно выбирать весовую функцию. Например, можно попробовать следующие варианты:

- $w(d) = \frac{1}{(d+a)^b}$
- $w(d) = q^d, \quad 0 < q < 1$

Существует более общий подход к придумыванию функции весов, зависящих от расстояний, который основан на использовании так называемых ядер. Но этот подход сейчас не будет обсуждаться.

12.3. Метрики в kNN

12.3.1. Понятие метрики

Метрика является функцией, задающей расстояние в метрическом пространстве, и должна удовлетворять следующим аксиомам:

1. $\rho(x, y) \geq 0$, причем $\rho(x, y) = 0 \iff x = y$.
2. $\rho(x, y) = \rho(y, x)$,
3. $\rho(x, y) \leq \rho(x, z) + \rho(z, y)$.

Эти аксиомы не будут обсуждаться в рамках данного раздела, так как излагаемый далее материал носит исключительно прикладной характер.

Можно привести следующие примеры метрик:

- Евклидова метрика:

$$\rho(x, y) = \sqrt{\sum_{i=1}^n (x_i - y_i)^2}$$

- Манхэттенская метрика:

$$\rho(x, y) = \sum_{i=1}^n |x_i - y_i|$$

- Метрика Минковского (обобщение Евклидовой и Манхэттенской метрик):

$$\rho(x, y) = \left(\sum_{i=1}^n |x_i - y_i|^q \right)^{\frac{1}{q}}.$$

Наглядно демонстрируют структуру метрики изображения единичных окружностей в ней.

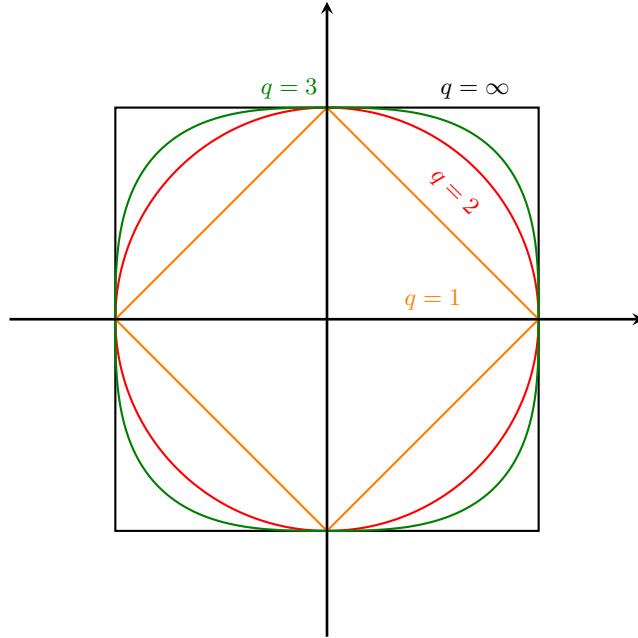


Рис. 12.7: Изображения единичной окружности, то есть множества точек, удаленных на расстояние 1 от начала координат, в различных метриках.

12.3.2. Функции близости

Часто, особенно в задачах анализа текста, используется так называемая косинусная мера, которая представляет собой косинус угла между векторами:

$$\text{similarity} = \cos \theta = \frac{x \cdot y}{\|x\| \cdot \|y\|} = \frac{\sum_{i=1}^n x_i y_i}{\sqrt{\sum_{i=1}^n x_i^2} \sqrt{\sum_{i=1}^n y_i^2}}$$

Причем косинусная мера — не расстояние, а функция близости, то есть такая функция, которая тем больше, чем больше объекты друг на друга похожи.

В рекомендательных системах используется коэффициент корреляции r . Он также может быть использован как функция близости и похож на косинусную меру:

$$r = \frac{\sum_{i=1}^n ((x_i - \bar{x})(y_i - \bar{y}))}{\sqrt{\sum_{i=1}^n (x_i - \bar{x})^2} \sqrt{\sum_{i=1}^n (y_i - \bar{y})^2}}$$

Конечно, существует и много других функций близости, которые в разной степени учитывают разные различия между векторами:

- Скалярное произведение: $\sum x_i y_i$
- Коэффициент Дайса: $\frac{2 \sum x_i y_i}{\sum x_i^2 + \sum y_i^2}$
- Косинусная мера: $\frac{\sum x_i y_i}{\sqrt{\sum x_i^2} \sqrt{\sum y_i^2}}$
- Коэффициент Жаккара: $\frac{\sum x_i y_i}{\sum x_i^2 + \sum y_i^2 - \sum x_i y_i}$

12.4. Проклятие размерности

В этом разделе будут рассмотрены основные особенности применения понятия расстояния в случае пространств большой размерности. Часто пространства признаков в задачах машинного обучения как раз и относятся к этому случаю.

12.4.1. Небольшие отличия в большом числе координат

Пусть даны три точки, радиус-векторы которых:

$$\begin{aligned}x_1 &= (a_1, a_2, \dots, a_N), \\x_2 &= (a_1 + \varepsilon, a_2 + \varepsilon, \dots, a_N + \varepsilon), \\x_3 &= (a_1, a_2 + \Delta, \dots, a_N).\end{aligned}$$

Вектор x_2 отличается от вектора x_1 незначительно в каждой координате, а вектор x_3 — только в одной, но существенно. При этом расстояние от первой точки до второй и расстояние от первой до третьей могут совпадать.

Другими словами, когда признаков очень много, незначительные различия в каждом признаке могут значить больше, чем одно большое различие в одном. Такое поведение не всегда является желаемым.

12.4.2. Почти одинаковые расстояния

Когда количество объектов сравнимо с количеством признаков, может возникнуть ситуация, что расстояния между двумя любыми объектами будет почти одинаковым.

Действительно, на плоскости три точки, равноудаленные друг от друга, образуют вершины треугольника, в трехмерном пространстве четыре равноудаленные друг от друга, точки являются вершинами тетраэдра, а в N -мерном пространстве можно выбрать $N + 1$ точку так, что расстояние между любыми двумя было одинаковым.

12.4.3. Экспоненциальный рост необходимых данных

Пусть X — вектор в признаковом пространстве из N бинарных признаков, например:

$$X = (0, 0, 1, 0, 1, 1, \dots, 1).$$

Всего в этом пространстве 2^N различных векторов, а значит размер обучающей выборки, необходимый, чтобы покрыть все возможные комбинации эти признаков будет также порядка 2^N .

Другими словами, количество необходимых данных с ростом размерности пространства экспоненциально увеличивается.

У данного факта есть красавая геометрическая иллюстрация. Пусть в N -мерном пространстве дан куб с ребром 1 и меньший куб, длина ребер которого равна $\ell < 1$. Меньший куб вложен в больший таким образом, что они имеют общую вершину и их грани попарно параллельны. Доля объема меньшего куба от объема большего выражается формулой:

$$\frac{v}{V} = \ell^N \rightarrow 0, \quad N \rightarrow \infty,$$

где v и V — объемы меньшего и большего кубов соответственно.

Пусть $\ell = \frac{1}{2}$. Если ставится задача описать положение какой-либо точки из большого куба с точностью до размеров маленького, то при увеличении размерности пространства и сохранении линейных размеров кубов количество требуемых для этого данных растет экспоненциально.

12.5. Рекомендации фильмов с помощью kNN

12.5.1. Задача

В данном разделе решается задача построения рекомендательной системы с помощью метода ближайших соседей. А именно необходимо построить рекомендации фильмов на основе истории пользовательских оценок.

Можно представить себе таблицу «пользователь-фильм», в которой какие-то значения заполнены, но не все. Таким образом, необходимо научиться прогнозировать значения, отсутствующие в данной таблице.

12.5.2. User-based подход

Один из возможных подходов к задаче заключается в том, что среди пользователей ищутся наиболее похожие на того пользователя, для которого делается прогноз. Этим пользователям в соответствие ставятся веса: тем, кто более похож, вес устанавливается больше, и так далее. В качестве прогноза для исходного пользователя

	Пила	Улица Вязов	Ванильное небо	1 + 1
Маша	5	4	1	2
Юля		5	2	
Вова			3	5
Коля	3		4	5
Петя				4
Ваня		5	3	3

Таб.: Пример возможной таблицы «пользователь-фильм».

указываются усредненные с учетом весовых коэффициентов оценки фильмов этих наиболее похожих на него пользователей.

Описанный выше прогноз называется user-based. Аналогично можно рассмотреть item-based подход, где сначала выбирается интересующий фильм, ищутся похожие на него фильмы и так далее. Далее для определенности речь будет идти только про user-based подход.

12.5.3. Похожесть пользователей

В качестве меры похожести $w_{i,j}$ двух пользователей можно использовать коэффициент корреляции:

$$w_{i,j} = \frac{\sum_a (r_{i,a} - \bar{r}_i)(r_{j,a} - \bar{r}_j)}{\sqrt{\sum_a (r_{i,a} - \bar{r}_i)^2} \sqrt{\sum_a (r_{j,a} - \bar{r}_j)^2}},$$

где $\bar{r}_i = \frac{1}{N_i} \sum_a r_{i,a}$ — средние оценки i -го пользователя, N_i — количество просмотренных им фильмов. Суммирование ведется только по тем фильмам, которые смотрели оба пользователя.

12.5.4. Прогнозирование рейтинга

Теперь можно спрогнозировать рейтинг фильма как средний рейтинг с добавленной к нему взвешенной суммой рейтингов других пользователей:

$$\hat{r}_{i,a} = \bar{r}_i + \frac{\sum_j (r_{j,a} - \bar{r}_j) w_{i,j}}{\sum_j |w_{i,j}|}.$$

Если количество пользователей в системе велико, достаточно производить суммирование только по k ближайших соседей к пользователю, для которого дается оценка:

$$\hat{r}_{i,a} = \bar{r}_i + \frac{\sum_{j \in kNN(i)} (r_{j,a} - \bar{r}_j) w_{i,j}}{\sum_{j \in kNN(i)} |w_{i,j}|}.$$

Таким образом метод kNN может быть адаптирован к задаче рекомендации.

12.6. Метод опорных векторов (SVM)

Давайте познакомимся поближе с еще одним методом, который не вошел в основную часть курса, а именно с методом опорных векторов.

12.6.1. Метод опорных векторов

Конечно, в каком-то виде вы с ним уже знакомы. Это просто линейный классификатор

$$a(x) = \text{sign}(\langle w, x \rangle - w_0),$$

использующий кусочно-линейную функцию потерь

$$L(M_i) = \max\{0, 1 - M_i\} = (1 - M_i)_+$$

и L_2 -регуляризатор:

$$\sum_{i=1}^{\ell} \underbrace{L(M_i)}_{\text{Функция потерь}} + \underbrace{\gamma \|w\|^2}_{\text{Квадратичный регуляризатор}} \rightarrow \min_w.$$

Но на самом деле метод был придуман не из общего вида линейных классификаторов и не из обобщения с функциями потерь и регуляризаторами. Он был придуман из других довольно простых соображений.

12.6.2. Разделяющая полоса в случае линейно разделимой выборки

Пусть для простоты рассматривается задача бинарной классификации и некоторая линейно разделимая выборка. Выборка называется линейно разделимой, если в пространстве признаков существует такая гиперплоскость, что объекты разных классов будут находиться по разные стороны от этой плоскости.

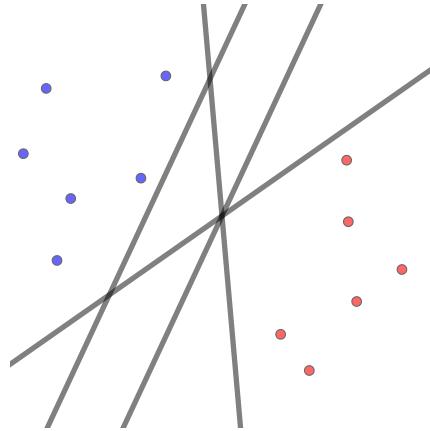


Рис. 12.8: Разделяющая гиперплоскость не единственна.

При этом гиперплоскость может быть проведена не единственным образом и возникает задача отыскания оптимальной разделяющей гиперплоскости.

Пусть разделяющая гиперплоскость существует и задается уравнением $\langle w, x \rangle - w_0 = 0$. Можно выбрать две параллельные ей и расположенные по разные стороны от нее гиперплоскости так, чтобы между ними не было объектов выборки, а расстояние между ними было максимальным. В таком случае каждая из двух получившихся граничных плоскостей будет «приставлена» к соответствующему классу.

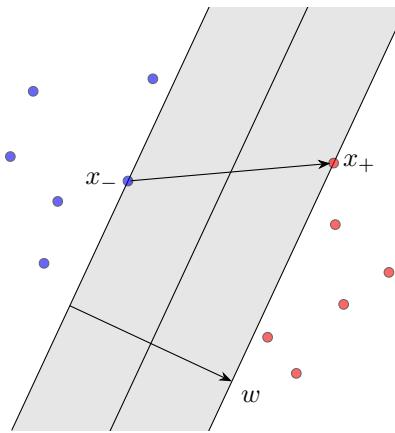


Рис. 12.9: Разделяющая полоса в случае линейно разделимой выборки

Поскольку уравнение плоскости можно умножать на ненулевое число без изменения соответствующей плоскости, всегда можно выбрать (отнормировать) w и w_0 таким образом, чтобы уравнения граничных плоскостей имели вид:

$$\langle w, x \rangle - w_0 = \pm 1.$$

Это условие нормировки можно также сформулировать следующим образом:

$$\min_{i=1, \dots, \ell} y_i(\langle w, x \rangle - w_0) = 1.$$

На каждой из двух граничных плоскостей будет лежать как минимум один объект из соответствующего класса (иначе расстояние между плоскостями можно увеличить). Пусть x_+ и x_- — два таких вектора, лежащие на построенных плоскостях и принадлежащие соответствующим классам.

Тогда для ширины разделяющей полосы будет справедливо выражение (как это следует из аналитической геометрии):

$$\left\langle (x_+ - x_-), \frac{w}{\|w\|} \right\rangle = \frac{2}{\|w\|}$$

Правая часть равенства получена в предположении, что используется описанная выше нормировка.

Теперь можно поставить задачу построения такой разделяющей гиперплоскости, что расстояние между соответствующими ей граничными плоскостями будет максимальным:

$$\begin{cases} \langle w, w \rangle \rightarrow \min, \\ y_i(\langle w, x \rangle - w_0) \geq 1, \quad i = 1, \dots, \ell. \end{cases}$$

12.6.3. Случай линейно неразделимой выборки

Поскольку в случае линейно неразделимой выборки по определению любой линейный классификатор будет ошибаться, условие $y_i(\langle w, x \rangle - w_0) \geq 1$ не может быть выполнено для всех i . Естественным обобщением задачи построения оптимальной гиперплоскости на случай линейно неразделимой выборки является введение ошибок $\xi_i \geq 0$ алгоритма и штрафов за эти ошибки в минимизируемую функцию следующим образом:

$$\begin{cases} \frac{1}{2} \langle w, w \rangle + C \sum_{i=1}^{\ell} \xi_i \rightarrow \min_{w, w_0, \xi}, \\ y_i(\langle w, x \rangle - w_0) \geq 1 - \xi_i, \quad i = 1, \dots, \ell, \\ \xi_i \geq 0, \quad i = 1, \dots, \ell. \end{cases}$$

Множитель $1/2$ был введен для удобства, а C задает размер штрафа за ошибки.

12.6.4. Оптимационная задача для метода опорных векторов

Получившаяся оптимационная задача:

$$\begin{cases} \frac{1}{2} \langle w, w \rangle + C \sum_{i=1}^{\ell} \xi_i \rightarrow \min_{w, w_0, \xi}, \\ y_i(\langle w, x \rangle - w_0) \geq 1 - \xi_i, \quad i = 1, \dots, \ell, \\ \xi_i \geq 0, \quad i = 1, \dots, \ell \end{cases}$$

является оптимационной задачей в методе опорных векторов (SVM) и непосредственно связана с задачей линейной классификации. Действительно, поскольку $M_i = y_i(\langle w, x \rangle - w_0)$ — отступ на i -ом объекте выборки:

$$y_i(\langle w, x \rangle - w_0) \geq 1 - \xi_i \implies \xi_i \geq 1 - M_i.$$

Учитывая также условие $\xi_i \geq 0$, можно получить:

$$\begin{cases} \frac{1}{2} \langle w, w \rangle + C \sum_{i=1}^{\ell} \xi_i \rightarrow \min_{w, w_0, \xi}, \\ \xi_i \geq \max\{0, 1 - M_i\}, \quad i = 1, \dots, \ell, \end{cases}$$

При фиксированных w и w_0 задача оптимизации по ξ имеет следующий вид:

$$\sum_{i=1}^{\ell} \xi_i \rightarrow \min_{\xi}, \quad \text{при условии } \xi_i \geq \max\{0, 1 - M_i\}, \quad i = 1, \dots, \ell,$$

а ее решением будет $\xi_i = \max\{0, 1 - M_i\} = (1 - M_i)_+$.

Теперь можно вернуться к общей задаче минимизации и переписать ее виде:

$$Q(w, w_0) = \sum_{i=1}^{\ell} (1 - M_i(w, w_0))_+ + \frac{1}{2C} \|w\|^2 \rightarrow \min_{w, w_0}.$$

Последнее выражение называется безусловной оптимационной задачей в SVN. В такой формулировке отчетливо видно и функцию потерь, и L_2 регуляризатор.

12.7. Ядра в методе опорных векторов (Kernel trick)

Пока ансамбли решающих деревьев не набрали своей популярности, SVM очень часто использовали даже в тех задачах, где разделяющая поверхность не похожа на линейную.

12.7.1. Добавление новых признаков и kernel trick

Чтобы применять SVN в нелинейном случае, строилось спрямляющее пространство. В основе этого лежит очень простая и очень красивая идея: если в каком-то исходном пространстве признаков классы не являются линейно разделимыми, то может быть можно отобразить это пространство признаков в какое-то новое, в котором классы уже будут линейно разделимы.

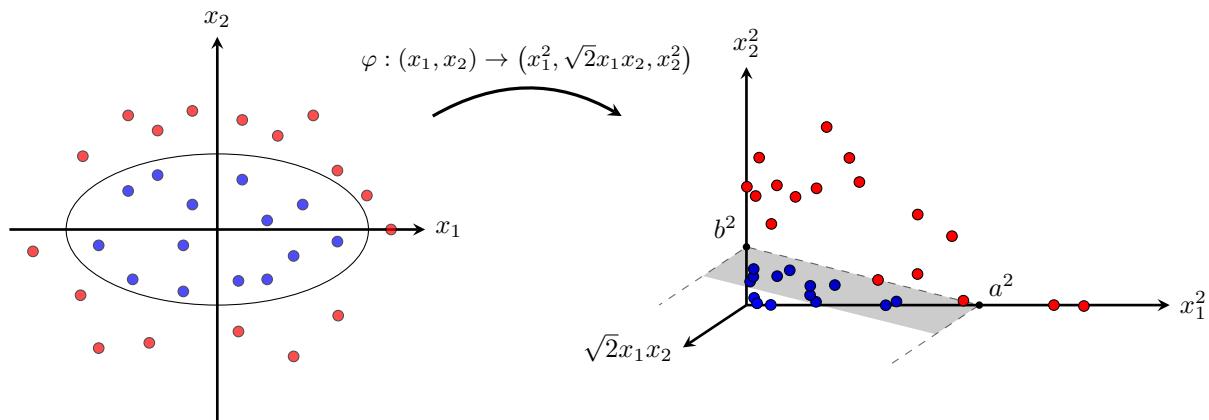


Рис. 12.10: Пример построения спрямляющего пространства.

Не обязательно задавать это отображение явно, так как в SVM везде фигурирует только скалярное произведение вида $\langle w, x \rangle$.

Пусть $\varphi(x)$ — спрямляющее отображение, тогда, чтобы записать SVM в спрямляющем пространстве, необходимо во всех формулах сделать следующие подстановки:

$$x \rightarrow \varphi(x), \quad w \rightarrow \varphi(w), \quad \langle w, x \rangle \rightarrow \langle \varphi(w), \varphi(x) \rangle.$$

Тогда метод SVM может быть сформулирован в исходном пространстве, если в качестве скалярного произведения использовать, возможно, нелинейную симметричную функцию

$$K(w, x) = \langle \varphi(w), \varphi(x) \rangle$$

и таким образом получать нелинейную разделяющую поверхность. Эта идея называется в англоязычной литературе *kernel trick*.

12.7.2. Линейное ядро

В простейшем случае ядро совпадает со скалярным произведением:

$$K(w, x) = \langle w, x \rangle.$$

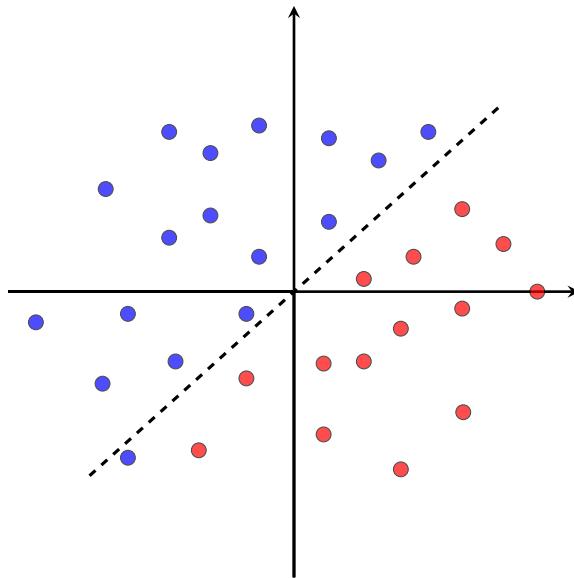


Рис. 12.11: Линейное ядро

Следует отметить, что линейное ядро в некоторых задачах — самый лучший выбор, например в задачах классификации текстов, и не стоит выкидывать его из рассмотрения.

12.7.3. Полиномиальное ядро

Другой пример — это полиномиальное ядро:

$$K(w, x) = (\langle w, x \rangle + r)^d.$$

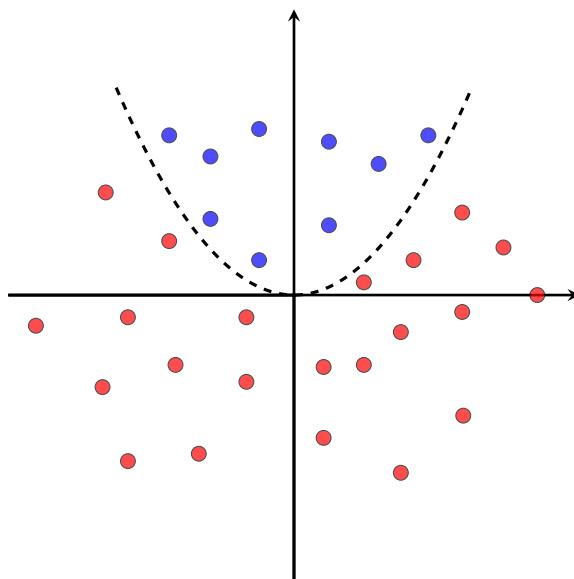


Рис. 12.12: Полиномиальное ядро

Полиномиальное ядро получается, если в качестве спрямляемого пространство выступает пространство многочленов не выше определенной степени.

12.7.4. Радиальное ядро

И другое часто используемое ядро — это радиальное ядро:

$$K(w, x) = \exp(-\gamma \|w - x\|^2).$$

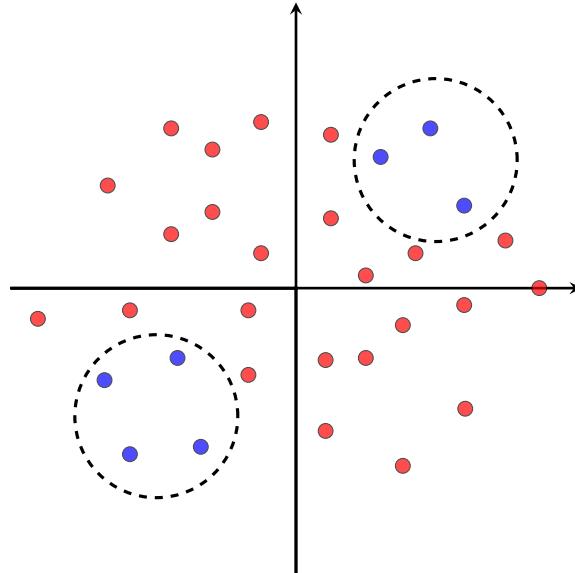


Рис. 12.13: Радиальное ядро

Поскольку радиальное ядро выражается через евклидово расстояние, будут проявляться основные проблемы метрических алгоритмов, в том числе проклятие размерности. Именно поэтому не стоит применять это ядро, если признаков действительно очень-очень много.

Но так или иначе, оно позволяет строить очень сложные границы классов. Спрямляющее пространство, которое соответствует данному ядру, является бесконечномерным.

12.7.5. Ядра и библиотеки

SVM реализован в различных библиотеках: какие-то (например LibSVM) поддерживают различные ядра, но некоторые (например LibLinear) — только линейное ядро. Причем, если на практике потребуется применить SVM с линейным ядром, лучше использовать LibLinear, который лучше оптимизирован для вычислений с линейным ядром, а не LibSVM с указанием линейного ядра.

Scikit-learn же просто предоставляет удобный доступ к LibSVM и LibLinear, поэтому все сказанное выше применимо и здесь: LinearSVC поддерживает только линейное ядро и оптимизирован для этого случая, а SVC поддерживает различные ядра.

Урок 13

Теорема Байеса в машинном обучении

13.1. Теорема Байеса

13.1.1. Ключевые понятия из теории вероятности

Пусть x и y — взаимозависимые случайные величины, тогда условная плотность на y при условии x по определению равна отношению совместной плотности распределения $p(x, y)$ к безусловной, или маргинальной, плотности $p(x)$:

$$p(y|x) = \frac{p(x, y)}{p(x)}.$$

Тогда:

$$p(y|x)p(x) = p(x, y) = p(x|y)p(y).$$

Отсюда следует формула обращения условной плотности:

$$p(x|y) = \frac{p(y|x)p(x)}{p(y)}.$$

Если взять интеграл от обоих частей этого выражения:

$$1 = \int p(x|y) = \int \frac{p(y|x)p(x)}{p(y)} dx = \frac{1}{p(y)} \int p(y|x)p(x)dx.$$

Отсюда следует, что безусловная (маргинальная) плотность распределения на y :

$$p(y) = \int p(y|x)p(x)dx$$

Это свойство часто называют правилом суммирования вероятностей.

Из правила суммирования вероятностей и правила произведения вероятностей следует знаменитая теорема Байеса:

$$p(y|x) = \frac{p(x|y)p(y)}{\int p(x|y)p(y)dy}.$$

Теорема Байеса позволяет переходить от априорных распределений на неизвестную величину (в данном случае на y) к апостериорным распределениям на y при условии x :

$$\text{Posterior} = \frac{\text{Likelihood} \times \text{Prior}}{\text{Evidence}}$$

13.1.2. Геометрический смысл условного и безусловного распределений

Рассматривается двухмерная случайная величина (ее плотность распределения $p(x, y)$ изображена на графике), компоненты которой взаимонезависимы.

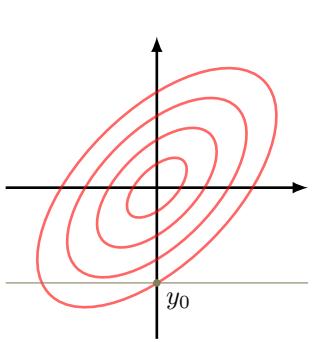


Рис. 13.1: Линии плотности случайной величины

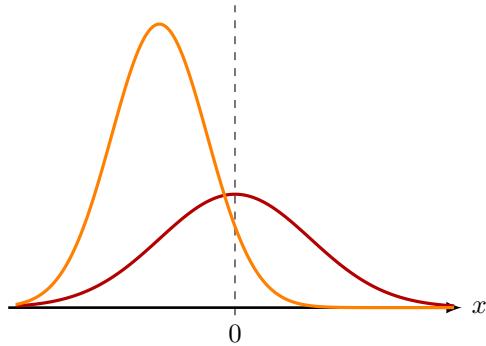


Рис. 13.2: Графики функций $p(x)$ (красный) и $p(x|y_0)$ (оранжевый).

Безусловное распределение $p(x)$ выражается как

$$p(x) = \int p(x, y) dy = \int p(x|y)p(y)dy$$

и является проекцией двухмерного графика на ось x . Очевидно, математическое ожидание x будет совпадать с матожиданием x в совместном вероятностном распределении. Если теперь стало известным значение $y = y_0$, то можно посчитать условное распределение на x при условии y_0 :

$$p(x|y_0) = \frac{p(x, y_0)}{p(y_0)},$$

этому распределению будут отвечать сечения двухмерного графика гиперплоскостью $y = y_0$. Следует обратить внимание, что у распределения $p(x|y_0)$, по сравнению с безусловным $p(x)$, изменилось математическое ожидание и уменьшилась дисперсия.

13.1.3. Метод максимального правдоподобия

Ну и, наконец, давайте вспомним ключевой метод статистического оценивания из классической статистики, известный как метод максимального правдоподобия.

Стандартная задача матстатистики, как известно, — оценить значение неизвестных параметров распределения по заданной выборке из этого распределения.

Пусть $X = (x_1, \dots, x_n)$ — выборка из независимых одинаково распределенных случайных величин:

$$x_i \sim p(x|\theta).$$

Плотность распределения $p(x|\theta)$ известна с точностью до параметров θ , которые необходимо оценить по известной выборке.

Метод максимального правдоподобия заключается в следующем. Сначала составляется функция правдоподобия выборки:

$$p(X, \theta) = \prod_{i=1}^n p(x_i|\theta).$$

Значения x_i являются известными, а значения θ выбираются таковыми, которые добавляют максимум функции правдоподобия:

$$\theta_{ML} = \operatorname{argmax}_{\theta} p(X, \theta) = \operatorname{argmax}_{\theta} \prod_{i=1}^n p(x_i|\theta) = \operatorname{argmax}_{\theta} \sum_{i=1}^n \ln p(x_i|\theta).$$

На практике обычно максимизируют не саму функцию правдоподобия, а ее логарифм, так как в этом случае произведение переходит в сумму логарифмов.

Для метода правдоподобия получен ряд интересных теоретических результатов:

- Оценка максимального правдоподобия является асимптотически несмешенной:

$$\mathbb{E}\theta_{ML} = \theta_* \quad \text{при} \quad n \gg 1.$$

- Оценка максимального правдоподобия является состоятельной:

$$\theta_{ML} \rightarrow \theta_* \quad \text{при} \quad n \rightarrow +\infty.$$

- Оценка максимального правдоподобия среди прочих несмешенных оценок обладает наименьшей возможной дисперсией, то есть обладает свойством эффективности.
- Оценки максимального правдоподобия асимптотически нормальны при больших n , то есть имеют нормальное распределение с математическим ожиданием, равным истинному значению, и ковариационной матрицей, связанной с матрицей информации Фишера.

Тем не менее большинство теоретических результатов, которые гарантируют корректность и оптимальность оценивания по методу максимального правдоподобия, получены при условии $\frac{n}{d} \gg 1$, где n — количество объектов в выборке, d — размерность θ . Если это условие не выполняется, многие результаты метода максимального правдоподобия перестают быть корректными.

13.2. Байесовский подход к теории вероятностей

13.2.1. Отличие байесовского подхода от классического

На самом деле ключевое различие между частотным подходом, который изучается в вузах, и байесовским подходом заключается в том, как трактовать случайность.

	Частотный	Байесовский
Интерпретация случайности	Объективная неопределенность	Субъективное незнание
Метод вывода	Метод максимального правдоподобия	Теорема Байеса
Оценки	θ_{ML}	$p(\theta X)$
Применимость	$\frac{n}{d} \gg 1$, где d — размерность θ .	Любые n

Таблица: Сравнение байесовского и классического подходов.

С точки зрения классического подхода, случайная величина — это величина, значение которой мы принципиально не можем предсказать, то есть некоторая объективная неопределенность.

В то же время с точки зрения байесовского подхода случайная величина на самом деле является детерминированным процессом, просто часть факторов, которые определяют исход этого процесса, для нас неизвестны. Именно поэтому мы и не можем предсказать конкретный исход данного испытания с данной случайной величиной.

Из этого сразу вытекают некоторые следствия. Например, с точки зрения байесовского подхода любую неизвестную величину можно интерпретировать как случайную и использовать аппарат теории вероятности, в частности, вводить на нее плотность распределения. При этом, коль скоро случайные величины кодируют субъективное незнание, у разных людей неопределенность на одну и ту же случайную величину может быть разная. Именно поэтому и плотности распределения на эту случайную величину будут отличаться для разных людей, обладающих разной информацией о факторах, влияющих на эту случайную величину.

С точки зрения классического подхода величины четко делятся на случайную и детерминированную и бесмысленно применять аппарат теории вероятности к детерминированным случайным величинам или параметрам. С точки зрения байесовского подхода все величины, значения которых неизвестны, можно интерпретировать как случайные и, соответственно, можно вводить плотность распределения и выполнять байесовский вывод.

Основным методом оценивания в классическом подходе является метод максимального правдоподобия. При байесовском подходе к статистике основным выводом является теорема Байеса. Соответственно, результатом оценивания в классическом подходе обычно являются точечные оценки, как правило, это оценки

максимального правдоподобия, либо реже — доверительные интервалы. При байесовском же подходе результатом вывода является апостериорное распределение на оцениваемые параметры. Метод максимального правдоподобия является оптимальным при $n \rightarrow \infty$, соответственно, большинство теорем в теории вероятности, которые обосновывают корректность применения этого метода, доказывают предположение, что объем выборки, по которой мы оцениваем неизвестный параметр, много больше 1.

В то же время байесовский подход можно использовать при любом объеме выборки, даже если объем выборки равен 0. В этом случае результатом байесовского вывода и апостериорного распределения просто будет являться априорное распределение. В то же время, если объем выборки, а именно отношение n к d , где n — это количество объектов, а d — это размерность оцениваемых параметров, много больше 1, результат байесовского вывода начинает стремиться к результату, оцениваемому с помощью метода максимального правдоподобия. Тем самым все теоретические гарантии, которые известны для метода максимального правдоподобия, применимы и к результату байесовского вывода.

13.2.2. Иллюстративный пример

Одним из преимуществ байесовского подхода является возможность объединения разных вероятностных моделей, которые отражают те или иные косвенные характеристики оцениваемой неизвестной величины.

Пусть y_1, \dots, y_m — несколько косвенных проявлений неизвестной величины x , для каждого из которых существует вероятностная модель $p_j(y_j|x)$, определяющая вероятность наблюдения того или иного значения y_j . Необходимо оценить x путем объединения информации из y_1, \dots, y_m .

Пусть $p(x)$ выражает исходные представления о возможных значениях x . Тогда после применения формулы Байеса можно получить апостериорное распределение на x при условии $y = y_1$:

$$p(x|y_1) = \frac{p_1(y_1|x)p(x)}{\int p_1(y_1|x)p(x)dx}.$$

При анализе результата второго измерения, которое может быть никак не связано с первым измерением и получено из совершенно другой вероятностной модели, нужно снова применить байесовский вывод, только теперь в качестве априорного распределения на x мы положим апостериорное распределение, полученное после измерения y_1 :

$$p(x|y_1, y_2) = \frac{p_2(y_2|x)p(x|y_1)}{\int p_2(y_2|x)p(x|y_1)dx}.$$

Действуя так m раз, можно получить апостериорное распределение на x при условии y_1, \dots, y_m , которое отражает максимум информации, которую можно извлечь в данном случае:

$$p(x|y_1, \dots, y_m) = \frac{p_m(y_m|x)p(x|y_1, \dots, y_{m-1})}{\int p_m(y_m|x)p(x|y_1, \dots, y_{m-1})dx}.$$

Если бы вместо апостериорных распределений использовались точечные оценки, это было бы похоже на положении слепых мудрецов из известной притчи, которые пытались изучать слона на основе различных тактильных ощущений. Как известно, в притче мудрецы не смогли прийти к единому мнению, в то же время, если бы они оперировали байесовским аппаратом и получали бы апостериорное распределение, скорее всего, они смогли бы прийти к мнению относительно того, что же они изучают.

13.3. Байесовские модели в задачах машинного обучения

Задачу машинного обучения можно интерпретировать как задачу восстановления зависимости между наблюдаемыми и скрытыми компонентами. При этом зависимость восстанавливается по обучающей выборке, в которой предполагается, что мы знаем и наблюдаемые, и скрытые компоненты.

Если оперировать в рамках вероятностного подхода или, более конкретно, в рамках байесовского подхода, то в качестве модели зависимости между наблюдаемыми и скрытыми компонентами используются совместные вероятностные распределения над наблюдаемыми, скрытыми компонентами и, если речь идет про байесовский подход, то еще и над параметрами, которые настраиваются в ходе процедуры обучения.

13.3.1. Задача линейной регрессии

В задаче линейной регрессии есть три группы переменных:

- $x \in \mathbb{R}^d$ — признаки объекта (наблюдаемые параметры),
- $t \in \mathbb{R}$ — целевая переменная (скрытые параметры),
- $w \in \mathbb{R}^d$ — веса линейной регрессии (эти параметры настраиваются в ходе процедуры обучения).

Тогда на байесовском языке такую модель линейной регрессии можно сформулировать в виде совместного вероятностного распределения $p(t, w|x)$ на t и w при условии x . Такие модели называются дискриминативными моделями. В противовес им рассматривают так называемые генеративные модели, в которых совместные распределения вводятся как на скрытые величины t и настраиваемые параметры w , так и на наблюдаемые параметры x . В рамках генеративных моделей помимо прочего возможно решать задачу по генерации новых объектов. В рамках же дискриминативных моделей класс задач, которые можно решать, ограничивается задачами прогноза скрытой компоненты t при условии наблюдаемой компоненты x .

В следующей дискриминативной модели:

$$p(t, w|x) = p(t, x|w)p(w) = \mathcal{N}(t|w^T x, \sigma^2) \mathcal{N}(w|0, I).$$

совместное распределение $p(t, w|x)$ было разложено по правилам произведения на функции правдоподобия $p(t, x|w)$ и априорное распределение $p(w)$ на w . В качестве функции правдоподобия и априорного распределения используются нормальные распределения с соответствующими параметрами.

Пусть задана обучающая выборка $(X, T) = (x_i, t_i)_{i=1}^n$ и выполняется поиск максимума апостериорного распределения:

$$\begin{aligned} w_{MP} &= \operatorname{argmax}_w p(w|X, T) = \operatorname{argmax}_w p(T|X, w)p(w) = \operatorname{argmax}_w \prod_{i=1}^n p(t_i|x_i, w)p(w) = \\ &= \operatorname{argmax}_w \left[\sum_{i=1}^n \ln p(t_i|x_i, w)p(w) + \ln p(w) \right] = \operatorname{argmin}_w \frac{1}{2\sigma^2} \sum_{i=1}^n (t_i - x_i^T w)^2 + \frac{1}{2} \|w\|^2 = \\ &= \operatorname{argmin}_w \left[\sum_{i=1}^n (t_i - x_i^T w)^2 + \sigma^2 \|w\|^2 \right]. \end{aligned}$$

Фактически был получен метод наименьших квадратов с L_2 -регуляризатором. Таким образом, известный метод наименьших квадратов с L_2 -регуляризатором может быть переформулирован на языке байесовских моделей и соответствует достаточно простой вероятностной модели, в которую было введено гауссовское априорное распределение с нулевым матожиданием на веса линейной регрессии.

13.3.2. Преимущества байесовских моделей в машинном обучении

Первое преимущество состоит в возможности построения сложных вероятностных моделей из более простых. Это становится возможным благодаря тому, что результат байесовского вывода в одной модели (то есть апостериорное распределение) можно использовать в качестве априорного распределения в следующей вероятностной модели. Тем самым происходит зацепление разных вероятностных моделей.

Еще одним преимуществом байесовских методов является возможность обработки массива данных, которые поступают последовательно. В самом деле, используя апостериорное распределение в качестве априорного при поступлении новой порции данных, можно легко произвести обновление апостериорного распределения без необходимости повторного обучения модели с нуля. При использовании точечных оценок нужно было бы заново обучать модель.

Еще одним преимуществом байесовских методов является возможность использования априорного распределения, которое предотвращает излишнюю настройку неизвестных параметров под обучающую выборку. Это в свою очередь позволяет избежать эффекта переобучения, который часто свойственен даже задачам, в которых присутствует гигантский объем обучающих выборок, но в ситуации, когда и количество настраиваемых параметров тоже достаточно велико. Благодаря использованию априорных распределений можно регуляризовать модель машинного обучения и предотвращать эффект переобучения.

Наконец, одним из ключевых достоинств байесовских методов является возможность работы с не полностью размеченными, частично размеченными, а то и вовсе не размеченными обучающими выборками. То есть в ситуациях, когда в обучающих выборках известна наблюдаемая компонента, а скрытая компонента известна не для всех объектов, либо для многих объектов известно не точное значение скрытой компоненты, а лишь некоторое допустимое подмножество скрытых компонент. Такие выборки называются частично размеченными. Оказывается, что байесовский формализм, байесовское моделирование позволяет абсолютно корректно работать с такими моделями и извлекать из них максимум имеющейся информации о неизвестных значениях параметров.