Машинное обучение I

Использовались метериалы курсов ФКН ВШЭ «Машинное обучение-1» и «Введение в анализ данных».

Содержание

1 Введение

1. Что такое объект, целевая переменная, признак, модель, функционал ошибки и обучение?

   - Пространство объектов $\mathbb{X}$ — это множество всех возможных точек размещения (e.g. все возможные расположения ресторанов).
   - Объект x — это то, для чего мы хотим делать предсказание (e.g. конкретное расположение ресторана).
   - Пространство ответов $\mathbb{Y}$ — это все возможные значения ответа.
   - Ответ y, или целевая переменная, — это то, что предсказываем (e.g. прибыль ресторана в течение первого года работы).
   - $X = \{(x_i, y_i)\}_{i=1}^{\ell}$ — *обучающая выбора*, где $x_1, \ldots, x_\ell$ — обучающие объекты, а ℓ — их количество. Особенность обучающих объектов состоит в том, что для них известны ответы $y_1, \ldots, y_\ell$.
   - Объекты являются некими абстрактными сущностями, с которыми компьютеры не умеют оперировать напрямую ⇒ для анализа данных необходимо описать объекты с помощью набора характеристик, которые называются *признаками*, или факторами. Вектор всех признаков объекта x_i называется *признаковым описанием* этого объекта $x_i = (x_{i1}, \ldots, x_{id})$.
   - Предположим, что мы собрали обучающую выборку и изобрели некоторое количество признаков. Результатом будет матрица «объекты-признаки» $X \in \mathbb{R}^{\ell \times d}$ (ℓ — число объектов, d — число признаков), в которой каждая строка содержит признаковое описание одного из обучающих объектов. Таким образом, строки в этой матрице соответствуют объектам, а столбцы — признакам.

     Нашей задачей является построение функции $a : \mathbb{X} \to \mathbb{Y}$, которая для любого объекта будет предсказывать ответ. Такая функция называется *алгоритмом* или *моделью*.

     Понятно, что нам подойдет далеко не каждый алгоритм — например, вряд ли мы извлечем какую-то выгоду из алгоритма $a(x) = 0$, который предсказывает нулевую прибыль для любого ресторана независимо от его признаков. Чтобы формализовать соответствие алгоритма нашим ожиданиям, нужно ввести *функционал качества* [1], измеряющий качество работы алгоритма. Отметим, что если функционал устроен так, что его следует минимизировать, то логичнее называть его *функционалом ошибки*. Крайне популярным функционалом в задаче регрессии является среднеквадратичная ошибка (mean squared error, MSE):

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

     Чем более маленькое значение этого функционала дает алгоритм, тем он лучше.

     В большинстве случаев функционалы представляют собой сумму ошибок на отдельных объектах обучающей выборки. Функция, измеряющая ошибку одного

[1] Достаточно часто употребляется термин «метрика качества», но он является не очень удачным, поскольку вызывает лишние ассоциации с метриками, обсуждаемыми в алгебре.

предсказания, называется *функцией потерь* $L : \mathbb{Y} \times \mathbb{Y} \to \mathbb{R}_+$ (в нашем случае $L(y, z) = (y - z)^2$).

Заметим, что именно функционал качества будет определять во всех дальнейших рассуждениях, какой алгоритм является лучшим. Если метрика выбрана неудачно и не соответствует бизнес-требованиям или особенностям данных, то все дальнейшие действия обречены на провал. Именно поэтому выбор базовой метрики является крайне важным этапом в решении любой задачи анализа данных. Она не обязательно должна обладать хорошими математическими свойствами (непрерывность, выпуклость, дифференцируемость и т.д.), но обязана отражать все важные требования к решению задачи.

Как только функционал качества зафиксирован, можно приступать к построению алгоритма $a(x)$. Как правило, для этого фиксируют некоторое *семейство алгоритмов* $\mathcal{A}$, и пытаются выбрать из него алгоритм, наилучший с точки зрения функционала [2].

В машинном обучении было изобретено большое количество семейств алгоритмов, и, наверное, самым простым и самым тщательно изученным среди них является семейство *линейных моделей*, которые дают предсказание, равное линейной комбинации признаков:

$$\mathcal{A} = \{a(x) = w_0 + w_1 x_1 + \cdots + w_d x_d \,|\, w_0, w_1, \ldots, w_d \in \mathbb{R}\},$$

где через x_i обозначается значение i-го признака у объекта x. Лучшая из таких моделей будет выбираться путем минимизации MSE-функционала:

$$\frac{1}{\ell} \sum_{i=1}^{\ell} \left(w_0 + \sum_{j=1}^{d} w_j x_{ij} - y_i \right)^2 \to \min_{w_0, w_1, \ldots, w_d}.$$

Через x_{ij} здесь обозначается значение j-го признака на i-м объекте. Процесс поиска оптимального алгоритма называется *обучением*. Если модель $a(x)$ дифференцируема по параметрам w, то можно искать лучший набор параметров с помощью градиентных методов — стартовать из случайной точки и двигать параметры в сторону наискорейшего убывания функционала ошибки (то есть в сторону антиградиента). Для выпуклых функционалов такой метод найдёт глобальный минимум; для невыпуклых функционалов есть лишь гарантии сходимости к локальному минимуму. При этом может возникнуть вопрос о том, как добиться сходимости к лучшему из локальных минимумов, и подходить к нему можно по-разному — например, с помощью выбора грамотного начального приближения или через выбор более сложного метода оптимизации. В то же время дифференцируемые модели, поддающиеся оптимизации, по сути, представляют собой последовательность несложных преобразований данных, и не факт, что такие модели смогут заменить сложные структуры данных или программы с ветвлениями и циклами. На сегодняшний день существует не так много видов недифференцируемых моделей, поддающихся эффективному обучению, но среди них есть очень успешные примеры — например, градиентный бустинг над решающими деревьями.

[2] «Пытаются выбрать» — потому что функционал может оказаться слишком сложным, не позволяющим точный поиск глобального минимума. В этом случае часто ограничиваются поиском локального экстремума.

2. Какие задачи встречаются в машинном обучении?

1) $\mathbb{Y} = \mathbb{R}$ — регрессия. Например, мы можем предсказывать, какова заработная плата у определенного человека.
2) $\mathbb{Y} = \{0, 1\}$ — бинарная классификация. Например, мы можем предсказывать, кликнет ли пользователь по рекламному объявлению, вернет ли клиент кредит в установленный срок, сдаст ли студент сессию, случится ли определенное заболевание с пациентом (на основе, скажем, его генома).
3) $\mathbb{Y} = \{1, \ldots, K\}$ — многоклассовая (multi-class) классификация. Примером может служить определение предметной области для научной статьи (математика, биология, психология и т.д.).
4) $\mathbb{Y} = \{0, 1\}^K$ — многоклассовая классификация с пересекающимися классами (multi-label classification). Примером может служить задача автоматического проставления тегов для ресторанов (логично, что ресторан может одновременно иметь несколько тегов).
5) Частичное обучение (semi-supervised learning) — задача, в которой для одной части объектов обучающей выборки известны и признаки, и ответы, а для другой только признаки. Такие ситуации возникают, например, в медицинских задачах, где получение ответа является крайне сложным (например, требует проведения дорогостоящего анализа).

Существует также обучение без учителя — класс задач, где ответы неизвестны или вообще не существуют, и требуется найти некоторые закономерности в данных лишь на основе признаковых описаний:

1) Кластеризация — задача разделения объектов на группы, обладающие некоторыми свойствами. Примером может служить кластеризация документов из электронной библиотеки или кластеризация абонентов мобильного оператора.
2) Оценивание плотности — задача приближения распределения объектов. Примером может служить задача обнаружения аномалий, в которой на этапе обучения известны лишь примеры «правильного» поведения оборудования (или, скажем, игроков на бирже), а в дальнейшем требуется обнаруживать случаи некорректной работы (соответственно, незаконного поведения игроков). В таких задачах сначала оценивается распределение «правильных» объектов, а затем аномальными объявляются все объекты, которых в рамках этого распределения получают слишком низкую вероятность.
3) Визуализация — задача изображения многомерных объектов в двумерном или трехмерном пространстве таким образом, чтобы сохранялось как можно больше зависимостей и отношений между ними.
4) Понижение размерности — задача генерации таких новых признаков, что их меньше, чем исходных, но при этом с их помощью задача решается не хуже (или с небольшими потерями качества, или лучше — зависит от постановки). К этой же категории относится задача построения латентных моделей, где требуется описать процесс генерации данных с помощью некоторого (как правило, небольшого) набора скрытых переменных. Примерами являются задачи

тематического моделирования и построения рекомендаций, которым будет посвящена часть курса.

3. Что такое вещественные (числовые), бинарные, категориальные признаки?

   Разработка признаков (feature engineering) для любой задачи является одним из самых сложных и самых важных этапов анализа данных.

   Признаки могут быть очень разными: бинарными, вещественными, категориальными (принимают значения из неупорядоченного множества), ординальными (принимают значения из упорядоченного множества), множествозначными (set-valued, значения являются подмножествами некоторого универсального множества). Признаки могут иметь сложную внутреннюю структуру: так, в качестве признака для конкретного человека в задаче предсказания его годового дохода может служить фотография.

4. Зачем нужна предобработка данных? Привидите примеры предобработки.

   1) Зачастую возникает потребность в *предобработке данных* до начала построения модели. Здесь может идти речь о некотором ряде манипуляций:
      - Некоторые модели хорошо работают только при выполнении определенных требований. Так, для линейных моделей крайне важно, чтобы признаки были *нормированными*, то есть измерялись в одной шкале. Примером способа нормировки данных является вычитание среднего и деление на дисперсию каждого столбца в матрице «объекты-признаки».
      - Бывает, что в выборку попадают *выбросы* — объекты, которые не являются корректными примерами из-за неправильно посчитанных признаков, ошибки сбора данных или чего-то еще. Их наличие может сильно испортить модель.
      - Некоторые признаки могут оказаться *шумовыми*, то есть не имеющими никакого отношения к целевой переменной и к решаемой задаче. Примером, скорее всего, может служить признак «фаза луны в день первого экзамена» в задаче предсказания успешности прохождения сессии студентом.

      Простейшая предобработка данных может радикально улучшить качество итоговой модели

   2) Примеры предобработки:
      - Категориальные признаки. Представим себе задачу определения стоимости квартиры по её характеристикам. Одним из важных признаков является район, в котором находится квартира. Этот признак является категориальным — его значения нельзя сравнивать между собой на больше/меньше, их нельзя складывать или вычитать. Непосредственно такие признаки нельзя использовать в линейных моделях, но есть достаточно распространённый способ их преобразования.
        Допустим, категориальный признак $f_j(x)$ принимает значения из множества $C = \{c_1, \ldots, c_m\}$. Заменим его на m бинарных признаков $b_1(x), \ldots, b_m(x)$, каждый из которых является индикатором одного из возможных категориальных значений:

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

Такой подход называется one-hot кодированием.
Отметим, что признаки $b_1(x), \ldots, b_m(x)$ являются линейно зависимыми, а именно для любого объекта выполнено:

$$b_1(x) + \cdots + b_m(x) = 1.$$

Чтобы избежать этого, можно выбрасывать один из бинарных признаков. Впрочем, такое решение имеет и недостатки — например, если на тестовой выборке появится новая категория, то её как раз можно закодировать с помощью нулевых бинарных признаков; при удалении одного из них это потеряет смысл.
Вернёмся к задаче про стоимость квартиры. Если мы применим линейную модель к данным после one-hot кодирования признака о районе (допустим, это $f(x)$), то получится такая формула:

$$a(x) = w_1[f(x) = c_1] + \cdots + w_m[f(x) = c_m] + \cdots + \{\text{другие признаки}\}.$$

Такая зависимость кажется логичной — каждый район задаёт некоторый базовый уровень стоимости (например, для района c_1 имеем базовую цену w_1), а остальные факторы корректируют его.

- Работа с текстами. Перейдём к предсказанию стоимости квартиры по её текстовому описанию. Есть простой способ кодирования, который называется *мешок слов (bag of words)*.
  Найдём все слова, которые есть в нашей выборке текстов, и пронумеруем их: $\{c_1, \ldots, c_m\}$. Будем кодировать текст m признаками $b_1(x), \ldots, b_m(x)$, где $b_j(x)$ равен количеству вхождений слова c_j в текст. Линейная модель над такими признаками будет иметь вид

  $$a(x) = w_1 b_1(x) + \cdots + w_m b_m(x) + \ldots,$$

  и такой вид тоже кажется разумным. Каждое вхождение слова c_j меняет прогноз стоимости на w_j. В самом деле, можно ожидать, что слово «престижный» скорее говорит о том, что квартира дорогая, а слово «плохой» вряд ли будут использовать при описании приличной квартиры.
- Бинаризация числовых признаков. Наконец, подумаем о предсказании стоимости квартиры по расстоянию до ближайшей станции метро x_j. Может оказаться, что самые дорогие квартиры расположены где-то в 5-10 минутах ходьбы от метро, а те, что ближе или дальше, стоят не так дорого. В этом случае зависимость целевой переменной от признака не будет линейной. Чтобы сделать линейную модель подходящей, мы можем бинаризовать признак. Для этого выберем некоторую сетку точек $\{t_1, \ldots, t_m\}$. Это может быть равномерная сетка между минимальным и максимальным значением признака или, например, сетка из эмпирических квантилей. Добавим сюда точки $t_0 = -\infty$ и $t_{m+1} = +\infty$. Новые признаки зададим как

  $$b_i(x) = [t_{i-1} < x_j \leqslant t_i], \quad i = 1, \ldots, m+1.$$

  Линейная модель над этими признаками будет выглядеть как

  $$a(x) = w_1[t_0 < x_j \leqslant t_1] + \cdots + w_{m+1}[t_m < x_j \leqslant t_{m+1}] + \ldots,$$

то есть мы найдём свой прогноз стоимости квартиры для каждого интервала расстояния до метро. Такой подход позволит учесть нелинейную зависимость между признаком и целевой переменной.

5. В чём заключается обобщающая способность алгоритма машинного обучения? К чему приводит её отсутствие? Что такое переобучение?

   Во время обучения модели очень важно следить за тем, чтобы не произошло *переобучения* (overfitting). Разберем это явление на примере. Допустим, что мы выбрали очень богатое семейство алгоритмов, состоящее из всех возможных функций: $\mathcal{A} = \{a : \mathbb{X} \to \mathbb{Y}\}$. В этом семействе всегда будет алгоритм, не допускающий ни одной ошибки на обучающей выборке, который просто запоминает ее:

$$a(x) = \begin{cases} y_i, & \text{если } x = x_i, \\ 0, & \text{если } x \notin X. \end{cases}$$

   Очевидно, что такой алгоритм нас не устраивает, поскольку для любого нового ресторана предскажет нулевую прибыль. Алгоритм оказался *переобученным* — он слишком сильно подогнался под обучающую выборку, не выявив никаких закономерностей в ней.

   Под переобученностью модели мы понимаем такую ситуацию, когда её качество на новых данных может быть существенно хуже качества на обучающей выборке. Действительно, при обучении мы требуем от модели лишь хорошего качества на обучающей выборке, и совершенно не очевидно, почему она должна при этом хорошо *обобщать* эти результаты на новые объекты.

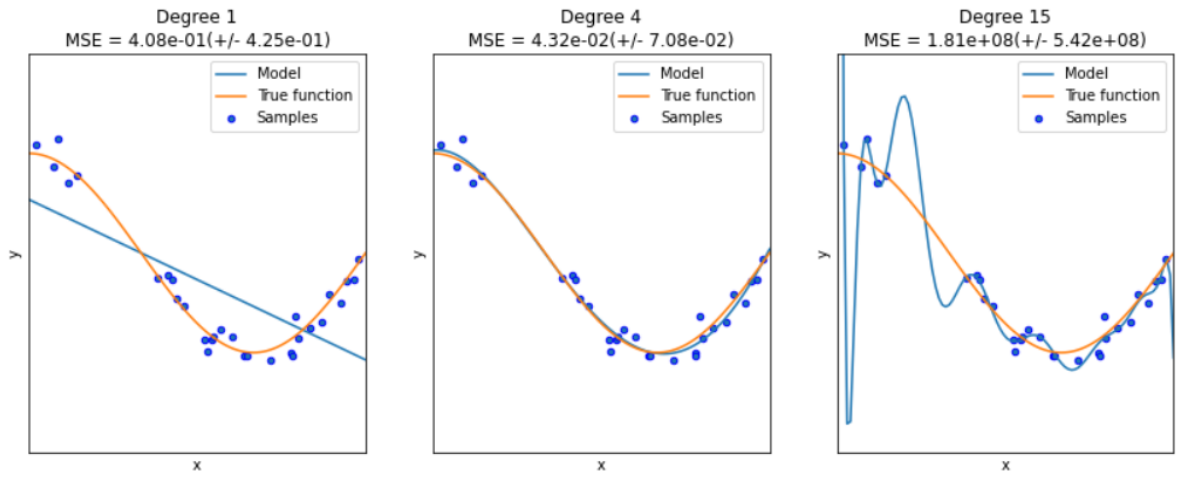

Рис. 1. Регрессионные кривые для признаковых наборов различной сложности.

Рассмотрим некоторую одномерную выборку, значения единственного признака x в которой генерируются равномерно на отрезке $[0, 1]$, а значения целевой переменной выбираются по формуле $y = \cos(1.5\pi x) + \mathcal{N}(0, 0.01)$, где $\mathcal{N}(\mu, \sigma^2)$ — нормальное распределение со средним μ и дисперсией σ^2. Попробуем восстановить зависимость с помощью линейных моделей над тремя наборами признаков: $\{x\}$, $\{x, x^2, x^3, x^4\}$ и $\{x, x^2, \ldots, x^{15}\}$. Соответствующие результаты представлены на рис. 1.

Видно, что при использовании признаков высоких степеней модель получает возможность слишком хорошо подстроиться под выборку, из-за чего становится непригодной для дальнейшего использования. Эту проблему можно решать многими способами — например, использовать более узкий класс моделей или штрафовать за излишнюю сложность полученной модели. Так, можно заметить, что у переобученной модели, полученной на третьем наборе признаков, получаются очень большие коэффициенты при признаках. Как правило, именно норма вектора коэффициентов используется как величина, которая штрафуется для контроля сложности модели. Такой подход называется *регуляризацией*.

6. Перечисли основные этапы решения задачи анализа данных.

   1) Постановка задачи;
   2) Выделение признаков;
   3) Формирование выборки;
   4) Выбор метрики качества;
   5) Предобработка данных;
   6) Построение модели;
   7) Оценивание качества модели.

7. Как правильно сравнивать различные методы машинного обучения?

   При сравнении различных методов машинного обучения принято сообщать относительное уменьшение ошибки. Рассмотрим два алгоритма a_1 и a_2 с долями правильных ответов r_1 и r_2 соответственно, причем $r_2 > r_1$. Относительным уменьшением ошибки алгоритма a_2 называется величина

   $$\frac{(1 - r_1) - (1 - r_2)}{1 - r_1}.$$

   Если доля ошибок была улучшена с 20% до 10%, то относительное улучшение составляет 50%. Если доля ошибок была улучшена с 50% до 25%, то относительное улучшение также равно 50%, хотя данный прирост кажется более существенным. Если же доля ошибок была улучшена с 0.1% до 0.01%, то относительное улучшение составляет 90%, что совершенно не соответствует здравому смыслу.

2 Линейная регрессия

§2.1 Вопросы для самопроверки

1. Запишите формулы для линейной модели регрессии и для среднеквадратичной ошибки.
2. Что такое коэффициент детерминации? Зачем он нужен и как интерпретировать его значения?
3. Чем отличаются функционалы MSE и MAE?
4. Опишите функционал MSLE.
5. Опишите функционалы MAPE и SMAPE.
6. Запишите среднеквадратичную ошибку в матричном виде. Какие недостатки есть у данного функционала?
7. Что такое градиент? Какое его свойство используется при минимизации функций?
8. Запишите формулу для одного шага градиентного спуска. Какие способы оценивания градиента вы знаете? Почему не всегда можно использовать полный градиентный спуск?
9. В чем идея Momentum, AdaGrad, RMSProp? Запишите их формулы.
10. Что такое отложенная выборка? Что такое кросс-валидация (скользящий контроль)? На что влияет количество блоков в кросс-валидации? Как ими пользоваться для выбора гиперпараметров?
11. Почему наличие линейно зависимых признаков представляет проблему при обучении линейной регрессии?
12. Что такое регуляризация? Запишите L1- и L2-регуляризаторы. Необходимо ли регуляризовать константый признак?
13. Что такое разреженные модели?
14. Как определить для линейной модели, какие признаки являются самыми важными?
15. Почему L1-регуляризация отбирает признаки?
16. Чем гиперпараметры отличаются от параметров? Что является параметрами и гиперпараметрами в линейных моделях и в решающих деревьях?
17. В чём заключается использование нелинейных признаков в линейных моделях? Для чего это нужно?
18. Что такое масштабирование признаков? Как его проводить? Зачем это нужно?

§2.2 Линейные модели

Такие модели сводятся к суммированию значений признаков с некоторыми весами:

$$a(x) = w_0 + \sum_{j=1}^{d} w_j x_j. \tag{2.1}$$

Параметрами модели являются *веса* или *коэффициенты* w_j. Вес w_0 также называется свободным коэффициентом или *сдвигом* (bias). Заметим, что сумма в формуле (2.1) является скалярным произведением вектора признаков на вектор весов. Воспользуемся этим и запишем линейную модель в более компактном виде:

$$a(x) = w_0 + \langle w, x \rangle, \tag{2.2}$$

где $w = (w_1, \ldots, w_d)$ — вектор весов.

Добавим к признаковому описанию каждого объекта $(d+1)$-й признак, равный единице. Вес при этом признаке как раз будет иметь смысл свободного коэффициента, и необходимость в слагаемом w_0 отпадёт:

$$a(x) = \langle w, x \rangle.$$

За счёт простой формы линейные модели достаточно быстро и легко обучаются, и поэтому популярны при работе с большими объёмами данных. Также у них мало параметров, благодаря чему удаётся контролировать риск переобучения и использовать их для работы с зашумлёнными данными и с небольшими выборками.

Сложно представить себе ситуацию, в которой мы берём данные, обучаем линейную модель и получаем хорошее качество работы. В линейной модели предполагается конкретный вид зависимости — а именно, что каждый признак линейно влияет на целевую переменную, и что целевая переменная не зависит от каких-либо комбинаций признаков. Вряд ли это будет выполнено по умолчанию, поэтому обычно данные требуют специальной подготовки, чтобы линейные модели оказались адекватными задаче (см. вопрос 4).

§2.3 Измерение ошибки в задачах регрессии

MSE. Основной способ измерить отклонение — посчитать квадрат разности:

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

Благодаря своей дифференцируемости эта функция наиболее часто используется в задачах регрессии. Основанный на ней функционал называется среднеквадратичным отклонением (mean squared error, MSE):

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

Отметим, что величина среднеквадратичного отклонения плохо интерпретируется, поскольку не сохраняет единицы измерения — так, если мы предсказываем цену

в рублях, то MSE будет измеряться в квадратах рублей. Чтобы избежать этого, используют корень из среднеквадратичной ошибки (root mean squared error, RMSE):

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

R^2. Среднеквадратичная ошибка подходит для сравнения двух моделей или для контроля качества во время обучения, но не позволяет сделать выводы о том, насколько хорошо данная модель решает задачу.

Например, MSE = 10 является очень плохим показателем, если целевая переменная принимает значения от 0 до 1, и очень хорошим, если целевая переменная лежит в интервале $(10000, 100000)$. В таких ситуациях вместо среднеквадратичной ошибки полезно использовать *коэффициент детерминации* (или коэффициент R^2):

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

где $\bar{y} = \frac{1}{\ell}\sum_{i=1}^{\ell} y_i$ — среднее значение целевой переменной. Коэффициент детерминации измеряет долю дисперсии, объяснённую моделью, в общей дисперсии целевой переменной. Фактически, данная мера качества — это нормированная среднеквадратичная ошибка. Если она близка к единице, то модель хорошо объясняет данные, если же она близка к нулю, то прогнозы сопоставимы по качеству с константным предсказанием.

MAE. Заменим квадрат отклонения на модуль:

$$L(y, a) = |a - y|$$

Соответствующий функционал называется средним абсолютным отклонением (mean absolute error, MAE):

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

Модуль отклонения не является дифференцируемым, но при этом менее чувствителен к выбросам. Квадрат отклонения, по сути, делает особый акцент на объектах с сильной ошибкой, и метод обучения будет в первую очередь стараться уменьшить отклонения на таких объектах. Если же эти объекты являются выбросами (то есть значение целевой переменной на них либо ошибочно, либо относится к другому распределению и должно быть проигнорировано), то такая расстановка акцентов приведёт к плохому качеству модели. Модуль отклонения в этом смысле гораздо более терпим к сильным ошибкам.

Приведём ещё одно объяснение того, почему модуль отклонения устойчив к выбросам, на простом примере. Допустим, все ℓ объектов выборки имеют одинаковые признаковые описания, но разные значения целевой переменной $y_1, \ldots, y_\ell$. В этом

случае модель должна на всех этих объектах выдать один и тот же ответ. Если мы выбрали MSE в качестве функционала ошибки, то получаем следующую задачу:

$$\frac{1}{\ell}\sum_{i=1}^{\ell}(a-y_i)^2 \to \min_a$$

Легко показать, что минимум достигается на среднем значении всех ответов:

$$a^*_{\text{MSE}} = \frac{1}{\ell}\sum_{i=1}^{\ell} y_i.$$

Если один из ответов на порядки отличается от всех остальных (то есть является выбросом), то среднее будет существенно отклоняться в его сторону.

Рассмотрим теперь ту же ситуацию, но с функционалом MAE:

$$\frac{1}{\ell}\sum_{i=1}^{\ell}|a-y_i| \to \min_a$$

Теперь решением будет медиана ответов:

$$a^*_{\text{MAE}} = \text{median}\{y_i\}_{i=1}^{\ell}.$$

Небольшое количество выбросов никак не повлияет на медиану — она существенно более устойчива к величинам, выбивающимся из общего распределения.

MSLE.

$$L(y,a) = (\log(a+1) - \log(y+1))^2$$

Соответствующий функционал называется среднеквадратичной логарифмической ошибкой (mean squared logarithmic error, MSLE). Данная метрика подходит для задач с неотрицательной целевой переменной. За счёт логарифмирования ответов и прогнозов мы скорее штрафуем за отклонения в порядке величин, чем за отклонения в их значениях. Также следует помнить, что логарифм не является симметричной функцией, и поэтому данная функция потерь штрафует заниженные прогнозы сильнее, чем завышенные.

MAPE и SMAPE. В задачах прогнозирования обычно измеряется относительная ошибка:

$$L(y,a) = \left|\frac{y-a}{y}\right|$$

Соответствующий функционал называется средней абсолютной процентной ошибкой (mean absolute percentage error, MAPE). Данный функционал часто используется в задачах прогнозирования. Также используется его симметричная модификация (symmetric mean absolute percentage error, SMAPE):

$$L(y,a) = \frac{|y-a|}{(|y|+|a|)/2}$$

§2.4 Обучение линейной регрессии

Чаще всего линейная регрессия обучается с использованием среднеквадратичной ошибки. В этом случае получаем задачу оптимизации (считаем, что среди признаков есть константный, и поэтому свободный коэффициент не нужен):

$$\frac{1}{\ell}\sum_{i=1}^{\ell}\left(\langle w, x_i\rangle - y_i\right)^2 \to \min_w$$

Эту задачу можно переписать в матричном виде. Если X — матрица «объекты-признаки», y — вектор ответов, w — вектор параметров, то приходим к виду

$$\frac{1}{\ell}\left\|Xw - y\right\|^2 \to \min_w, \tag{2.3}$$

где используется обычная L_2-норма. Если продифференцировать данный функционал по вектору w, приравнять к нулю и решить уравнение, то получим явную формулу для решения:

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

Безусловно, наличие явной формулы для оптимального вектора весов — это большое преимущество линейной регрессии с квадратичным функционалом. Но данная формула не всегда применима по ряду причин:

- Обращение матрицы — сложная операция с кубической сложностью от количества признаков. Если в выборке тысячи признаков, то вычисления могут стать слишком трудоёмкими. Решить эту проблему можно путём использования численных методов оптимизации.
- Матрица $X^T X$ может быть вырожденной или плохо обусловленной. В этом случае обращение либо невозможно, либо может привести к неустойчивым результатам. Проблема решается с помощью регуляризации.

Следует понимать, что аналитические формулы для решения довольно редки в машинном обучении. Если мы заменим MSE на другой функционал, то найти такую формулу, скорее всего, не получится. Желательно разработать общий подход, в рамках которого можно обучать модель для широкого класса функционалов.

Такой подход действительно есть для дифференцируемых функций — градиентный спуск. Оптимизационные задачи вроде (2.3) можно решать итерационно с помощью градиентных методов (или же методов, использующих как градиент, так и информацию о производных более высокого порядка).

§2.5 Градиентный спуск и оценивание градиента

Градиентом функции $f : \mathbb{R}^d \to \mathbb{R}$ называется вектор его частных производных:

$$\nabla f(x_1, \ldots, x_d) = \left(\frac{\partial f}{\partial x_j}\right)_{j=1}^{d}.$$

Известно, что градиент является направлением наискорейшего роста функции, а антиградиент (т.е. $-\nabla f$) — направлением наискорейшего убывания. Это ключевое свойство градиента, обосновывающее его использование в методах оптимизации.

Докажем данное утверждение. Пусть $v \in \mathbb{R}^d$ — произвольный вектор, лежащий на единичной сфере: $\|v\| = 1$. Пусть $x_0 \in \mathbb{R}^d$ — фиксированная точка пространства. Скорость роста функции в точке x_0 вдоль вектора v характеризуется производной по направлению $\frac{\partial f}{\partial v}$:

$$\frac{\partial f}{\partial v} = \frac{d}{dt} f(x_{0,1} + tv_1, \ldots, x_{0,d} + tv_d)|_{t=0}.$$

Из курса математического анализа известно, что данную производную сложной функции можно переписать следующим образом:

$$\frac{\partial f}{\partial v} = \sum_{j=1}^{d} \frac{\partial f}{\partial x_j} \frac{d}{dt} (x_{0,j} + tv_j) = \sum_{j=1}^{d} \frac{\partial f}{\partial x_j} v_j = \langle \nabla f, v \rangle.$$

Распишем скалярное произведение:

$$\langle \nabla f, v \rangle = \|\nabla f\| \|v\| \cos \varphi = \|\nabla f\| \cos \varphi,$$

где φ — угол между градиентом и вектором v. Таким образом, производная по направлению будет максимальной, если угол между градиентом и направлением равен нулю, и минимальной, если угол равен 180 градусам. Иными словами, производная по направлению максимальна вдоль градиента и минимальна вдоль антиградиента.

У градиента есть ещё одно свойство, которое пригодится нам при попытках визуализировать процесс оптимизации, — он ортогонален линиям уровня. Докажем это. Пусть x_0 — некоторая точка, $S(x_0) = \{x \in \mathbb{R}^d \mid f(x) = f(x_0)\}$ — соответствующая линия уровня. Разложим функцию в ряд Тейлора на этой линии в окрестности x_0:

$$f(x_0 + \varepsilon) = f(x_0) + \langle \nabla f, \varepsilon \rangle + o(\|\varepsilon\|),$$

где $x_0 + \varepsilon \in S(x_0)$. Поскольку $f(x_0 + \varepsilon) = f(x_0)$ (как-никак, это линия уровня), получим

$$\langle \nabla f, \varepsilon \rangle = o(\|\varepsilon\|).$$

Поделим обе части на $\|\varepsilon\|$:

$$\left\langle \nabla f, \frac{\varepsilon}{\|\varepsilon\|} \right\rangle = o(1).$$

Устремим $\|\varepsilon\|$ к нулю. При этом вектор $\frac{\varepsilon}{\|\varepsilon\|}$ будет стремится к касательной к линии уровня в точке x_0. В пределе получим, что градиент ортогонален этой касательной.

2.5.1 Градиентный спуск

Основное свойство антиградиента — он указывает в сторону наискорейшего убывания функции в данной точке. Соответственно, будет логично стартовать из некоторой

точки, сдвинуться в сторону антиградиента, пересчитать антиградиент и снова сдвинуться в его сторону и т.д. Запишем это более формально. Пусть $w^{(0)}$ — начальный набор параметров (например, нулевой или сгенерированный из некоторого случайного распределения). Тогда градиентный спуск состоит в повторении следующих шагов до сходимости:

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla Q(w^{(k-1)}). \tag{2.4}$$

Здесь под $Q(w)$ понимается значение функционала ошибки для набора параметров w.

Через η_k обозначается длина шага, которая нужна для контроля скорости движения. Можно делать её константной: $\eta_k = c$. При этом если длина шага слишком большая, то есть риск постоянно «перепрыгивать» через точку минимума, а если шаг слишком маленький, то движение к минимуму может занять слишком много итераций. Иногда длину шага монотонно уменьшают по мере движения — например, по простой формуле

$$\eta_k = \frac{1}{k}.$$

В пакете `vowpal wabbit`, реализующем настройку и применение линейных моделей, используется более сложная формула для шага в градиентном спуске:

$$\eta_k = \lambda \left(\frac{s_0}{s_0 + k} \right)^p,$$

где λ, s_0 и p — параметры (мы опустили в формуле множитель, зависящий от номера прохода по выборке). На практике достаточно настроить параметр λ, а остальным присвоить разумные значения по умолчанию: $s_0 = 1$, $p = 0.5$, $d = 1$.

Останавливать итерационный процесс можно, например, при близости градиента к нулю или при слишком малом изменении вектора весов на последней итерации.

Если функционал $Q(w)$ выпуклый, гладкий и имеет минимум w^*, то имеет место следующая оценка сходимости:

$$Q(w^{(k)}) - Q(w^*) = O(1/k).$$

2.5.2 Оценивание градиента.

Как правило, в задачах машинного обучения функционал $Q(w)$ представим в виде суммы ℓ функций:

$$Q(w) = \frac{1}{\ell} \sum_{i=1}^{\ell} q_i(w).$$

В таком виде, например, записан функционал в задаче (2.3), где отдельные функции $q_i(w)$ соответствуют ошибкам на отдельных объектах.

Проблема метода градиентного спуска (2.4) состоит в том, что на каждом шаге необходимо вычислять градиент всей суммы (будем его называть полным градиентом):

$$\nabla_w Q(w) = \frac{1}{\ell} \sum_{i=1}^{\ell} \nabla_w q_i(w).$$

Это может быть очень трудоёмко при больших размерах выборки. В то же время точное вычисление градиента может быть не так уж необходимо — как правило, мы делаем не очень большие шаги в сторону антиградиента, и наличие в нём неточностей не должно сильно сказаться на общей траектории. Опишем несколько способов оценивания полного градиента.

Оценить градиент суммы функций можно градиентом одного случайно взятого слагаемого:

$$\nabla_w Q(w) \approx \nabla_w q_{i_k}(w),$$

где i_k — случайно выбранный номер слагаемого из функционала. В этом случае мы получим метод *стохастического градиентного спуска* (stochastic gradient descent, SGD) [1]:

$$w^{(k)} = w^{(k-1)} - \eta_k \nabla q_{i_k}(w^{(k-1)}).$$

Для выпуклого и гладкого функционала может быть получена следующая оценка:

$$\mathbb{E}\left[Q(w^{(k)}) - Q(w^*)\right] = O(1/\sqrt{k}).$$

Таким образом, метод стохастического градиента имеет менее трудоемкие итерации по сравнению с полным градиентом, но и скорость сходимости у него существенно меньше.

Отметим одно важное преимущество метода стохастического градиентного спуска. Для выполнения одного шага в данном методе требуется вычислить градиент лишь одного слагаемого — а поскольку одно слагаемое соответствует ошибке на одном объекте, то получается, что на каждом шаге необходимо держать в памяти всего один объект из выборки. Данное наблюдение позволяет обучать линейные модели на очень больших выборках: можно считывать объекты с диска по одному, и по каждому делать один шаг метода SGD.

Можно повысить точность оценки градиента, используя несколько слагаемых вместо одного:

$$\nabla_w Q(w) \approx \frac{1}{n} \sum_{j=1}^{n} \nabla_w q_{i_{kj}}(w),$$

где i_{kj} — случайно выбранные номера слагаемых из функционала (j пробегает значения от 1 до n), а n — параметр метода, размер пачки объектов для одного градиентного шага. С такой оценкой мы получим метод mini-batch gradient descent, который часто используется для обучения дифференцируемых моделей.

В 2013 году был предложен метод *среднего стохастического градиента* (stochastic average gradient) [2], который в некотором смысле сочетает низкую сложность итераций стохастического градиентного спуска и высокую скорость сходимости полного градиентного спуска. В начале работы в нём выбирается первое приближение w^0, и инициализируются вспомогательные переменные z_i^0, соответствующие градиентам слагаемых функционала:

$$z_i^{(0)} = \nabla q_i(w^{(0)}), \qquad i = 1, \ldots, \ell.$$

На k-й итерации выбирается случайное слагаемое i_k и обновляются вспомогательные переменные:

$$z_i^{(k)} = \begin{cases} \nabla q_i(w^{(k-1)}), & \text{если } i = i_k; \\ z_i^{(k-1)} & \text{иначе.} \end{cases}$$

Иными словами, пересчитывается один из градиентов слагаемых. Оценка градиента вычисляется как среднее вспомогательных переменных — то есть мы используем все слагаемые, как в полном градиенте, но при этом почти все слагаемые берутся с предыдущих шагов, а не пересчитываются:

$$\nabla_w Q(w) \approx \frac{1}{\ell} \sum_{i=1}^{\ell} z_i^{(k)}.$$

Наконец, делается градиентный шаг:

$$w^{(k)} = w^{(k-1)} - \eta_k \frac{1}{\ell} \sum_{i=1}^{\ell} z_i^{(k)}.$$

Данный метод имеет такой же порядок сходимости для выпуклых и гладких функционалов, как и обычный градиентный спуск:

$$\mathbb{E}\left[Q(w^{(k)}) - Q(w^*)\right] = O(1/k).$$

Существует множество других способов получения оценки градиента. Например, это можно делать без вычисления каких-либо градиентов вообще [3] — достаточно взять случайный вектор u на единичной сфере и домножить его на значение функции в данном направлении:

$$\nabla_w Q(w) = Q(w + \delta u)u.$$

Можно показать, что данная оценка является несмещённой для сглаженной версии функционала Q.

В задаче оценивания градиента можно зайти ещё дальше. Если вычислять градиенты $\nabla_w q_i(w)$ сложно, то можно *обучить модель*, которая будет выдавать оценку градиента на основе текущих значений параметров. Этот подход был предложен для обучения глубинных нейронных сетей [4].

2.5.3 Модификации градиентного спуска

С помощью оценок градиента можно уменьшать сложность одного шага градиентного спуска, но при этом сама идея метода не меняется — мы движемся в сторону наискорейшего убывания функционала. Конечно, такой подход не идеален, и можно по-разному его улучшать, устраняя те или иные его проблемы. Мы разберём два примера таких модификаций — одна будет направлена на борьбу с осцилляциями, а вторая позволит автоматически подбирать длину шага.

Метод импульса (momentum). Может оказаться, что направление антиградиента сильно меняется от шага к шагу. Например, если линии уровня функционала сильно вытянуты, то из-за ортогональности градиента линиям уровня он будет менять направление на почти противоположное на каждом шаге. Такие осцилляции будут вносить сильный шум в движение, и процесс оптимизации займёт много итераций. Чтобы избежать этого, можно усреднять векторы антиградиента с нескольких предыдущих шагов — в этом случае шум уменьшится, и такой средний вектор будет указывать в сторону общего направления движения. Введём для этого вектор инерции:

$$h_0 = 0;$$
$$h_k = \alpha h_{k-1} + \eta_k \nabla_w Q(w^{(k-1)}).$$

Здесь α — параметр метода, определяющей скорость затухания градиентов с предыдущих шагов. Разумеется, вместо вектора градиента может быть использована его аппроксимация. Чтобы сделать шаг градиентного спуска, просто сдвинем предыдущую точку на вектор инерции:

$$w^{(k)} = w^{(k-1)} - h_k.$$

Заметим, что если по какой-то координате градиент постоянно меняет знак, то в результате усреднения градиентов в векторе инерции эта координата окажется близкой к нулю. Если же по координате знак градиента всегда одинаковый, то величина соответствующей координаты в векторе инерции будет большой, и мы будем делать большие шаги в соответствующем направлении.

AdaGrad и RMSprop. Градиентный спуск очень чувствителен к выбору длины шага. Если шаг большой, то есть риск, что мы будем «перескакивать» через точку минимума; если же шаг маленький, то для нахождения минимума потребуется много итераций. При этом нет способов заранее определить правильный размер шага — к тому же, схемы с постепенным уменьшением шага по мере итераций могут тоже плохо работать.

В методе AdaGrad предлагается сделать свою длину шага для каждой компоненты вектора параметров. При этом шаг будет тем меньше, чем более длинные шаги мы делали на предыдущих итерациях:

$$G_{kj} = G_{k-1,j} + (\nabla_w Q(w^{(k-1)}))_j^2;$$
$$w_j^{(k)} = w_j^{(k-1)} - \frac{\eta_t}{\sqrt{G_{kj} + \varepsilon}} (\nabla_w Q(w^{(k-1)}))_j.$$

Здесь ε — небольшая константа, которая предотвращает деление на ноль. В данном методе можно зафксировать длину шага (например, $\eta_k = 0.01$) и не подбирать её в процессе обучения. Отметим, что данный метод подходит для разреженных задач, в которых у каждого объекта большинство признаков равны нулю. Для признаков, у которых ненулевые значения встречаются редко, будут делаться большие шаги; если же какой-то признак часто является ненулевым, то шаги по нему будут небольшими.

У метода AdaGrad есть большой недостаток: переменная G_{kj} монотонно растёт, из-за чего шаги становятся всё медленнее и могут остановиться ещё до того,

как достигнут минимум функционала. Проблема решается в методе RMSprop, где используется экспоненциальное затухание градиентов:

$$G_{kj} = \alpha G_{k-1,j} + (1-\alpha)(\nabla_w Q(w^{(k-1)}))_j^2.$$

В этом случае размер шага по координате зависит в основном от того, насколько быстро мы двигались по ней на последних итерациях.

§2.6 Оценивание качества моделей

В вопросе 5, в первом разделе, в последнем примере, мы не можем обнаружить переобученность модели по обучающей выборке [3]. С другой стороны, если бы у нас были дополнительные объекты с известными ответами, то по ним заметить низкое качество модели было бы довольно легко.

На данной идее основан подход с *отложенной выборкой*. Имеющиеся размеченные данные (т.е. данные с известными ответами) разделяются на две части: обучающую и контрольную. На обучающей выборке, как это следует из названия, модель обучается, а на контрольной выборке проверяется её качество. Если значение функционала на контрольной выборке оказалось удовлетворительным, то можно считать, что модель смогла извлечь закономерности при обучении.

Использование отложенной выборки приводит к одной существенной проблеме: результат существенно зависит от конкретного разбиения данных на обучение и контроль. Мы не знаем, какое качество получилось бы, если бы объекты из данного контроля оказались в обучении. Решить эту проблему можно с помощью *кросс-валидации*. Размеченные данные разбиваются на k блоков $X_1, \ldots, X_k$ примерно одинакового размера. Затем обучается k моделей $a_1(x), \ldots, a_k(x)$, причём i-я модель обучается на объектах из всех блоков, кроме блока i. После этого качество каждой модели оценивается по тому блоку, который не участвовал в её обучении, и результаты усредняются:

$$\text{CV} = \frac{1}{k}\sum_{i=1}^{k} Q\left(a_i(x), X_i\right).$$

§2.7 Регуляризация

Мы упоминали, что для линейной регрессии если матрица X^TX не является обратимой, то с оптимизацией среднеквадратичной ошибки могут возникнуть некоторые трудности. Действительно, в ряде случаев (признаков больше чем объектов, коррелирующие признаки) оптимизационная задача $Q(w) \to \min$ может иметь бесконечное число решений, большинство которых являются переобученными и плохо работают на тестовых данных. Покажем это.

Пусть в выборке есть линейно зависимые признаки. Это по определению означает, что существует такой вектор v, что для любого объекта x выполнено $\langle v, x\rangle = 0$. Допустим, мы нашли оптимальный вектор весов w для линейного классификатора.

[3] Конечно, это можно было бы заметить по большим весам в модели, но связь между нормой весов и обобщающей способностью алгоритма неочевидна.

Но тогда классификаторы с векторами $w + \alpha v$ будут давать *точно такие же* ответы на всех объектах, поскольку

$$\langle w + \alpha v, x\rangle = \langle w, x\rangle + \alpha \underbrace{\langle v, x\rangle}_{=0} = \langle w, x\rangle.$$

Это значит, что метод оптимизации может найти решение со сколько угодно большими весами. Такие решения не очень хороши, поскольку классификатор будет чувствителен к крайне маленьким изменениям в признаках объекта, а значит, переобучен.

Мы уже знаем, что переобучение нередко приводит к большим значениям коэффициентов. Чтобы решить проблему, добавим к функционалу *регуляризатор*, который штрафует за слишком большую норму вектора весов:

$$Q_\alpha(w) = Q(w) + \alpha R(w).$$

Наиболее распространенными являются L_2 и L_1-регуляризаторы:

$$R(w) = \|w\|_2^2 = \sum_{i=1}^{d} w_i^2,$$

$$R(w) = \|w\|_1 = \sum_{i=1}^{d} |w_i|.$$

Коэффициент α называется параметром регуляризации и контролирует баланс между подгонкой под обучающую выборку и штрафом за излишнюю сложность. Разумеется, значение данного параметра следует подбирать под каждую задачу.

Отметим, что свободный коэффициент w_0 нет смысла регуляризовывать — если мы будем штрафовать за его величину, то получится, что мы учитываем некие априорные представления о близости целевой переменной к нулю и отсутствии необходимости в учёте её смещения. Такое предположение является достаточно странным. Особенно об этом следует помнить, если в выборке есть константный признак и коэффициент w_0 обучается наряду с остальными весами; в этом случае следует исключить слагаемое, соответствующее константному признаку, из регуляризатора.

Квадратичный (или L_2) регуляризатор достаточно прост в использовании в отличие от L_1-регуляризатора, у которого нет производной в нуле.

Обратим внимание на вид решения при использовании L_2-регуляризации вместе со среднеквадратичной ошибкой. В этом случае формулу для оптимального вектора весов можно записать в явном виде:

$$w = (X^T X + \alpha I)^{-1} X^T y.$$

Благодаря добавлению диагональной матрицы к $X^T X$ данная матрица оказывается положительно определённой, и поэтому её можно обратить. Таким образом, при использовании L_2 регуляризации решение всегда будет единственным.

§2.8 Разреженные модели

Использование L_1-регуляризатора приводит к обнулению части весов в модели. Обсудим подробнее, зачем это может понадобиться и почему так происходит.

Модели, в которых некоторые веса равны нулю, называют *разреженными*, поскольку прогноз в них зависит лишь от части признаков. Потребность в таких моделях можно возникнуть по многим причинам. Несколько примеров:

1) Может быть заведомо известно, что релевантными являются не все признаки. Очевидно, что признаки, которые не имеют отношения к задаче, надо исключать из данных, то есть производить *отбор признаков*. Есть много способов решения этой задачи, и L_1-регуляризация — один из них.

2) К модели могут выдвигаться ограничения по скорости построения предсказаний. В этом случае модель должна зависеть от небольшого количества наиболее важных признаков, и тут тоже оказывается полезной L_1-регуляризация.

3) В обучающей выборке объектов может быть существенно меньше, чем признаков (так называемая «проблема $N \ll p$»). Поскольку параметров линейной модели при этом тоже больше, чем объектов, задача обучения оказывается некорректной — решений много, и сложно выбрать из них то, которое обладает хорошей обобщающей способностью. Решить эту проблему можно путём внедрения в процесс обучения априорного знания о том, что целевая переменная зависит от небольшого количества признаков. Такая модификация как раз может быть сделана с помощью L_1-регуляризатора.

L_1-регуляризатор имеет интересную особенность: его использование приводит к занулению части весов. Этому есть несколько объяснений.

Угловые точки. Можно показать, что если функционал $Q(w)$ является выпуклым, то задача безусловной минимизации функции $Q(w) + \alpha\|w\|_1$ эквивалентна задаче условной оптимизации

$$\begin{cases} Q(w) \to \min\limits_{w} \\ \|w\|_1 \leqslant C \end{cases}$$

для некоторого C.

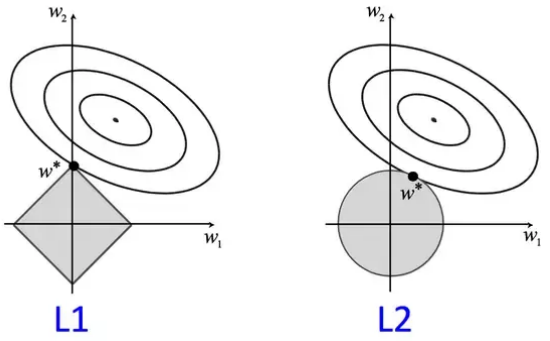

Рис. 2. Линии уровня функционала качества, а также ограничения, задаваемые L_1 и L_2-регуляризаторами.

На рис. 2 изображены линии уровня функционала $Q(w)$, а также множество, определяемое ограничением $\|w\|_1 \leqslant C$. Решение определяется точкой пересечения

допустимого множества с линией уровня, ближайшей к безусловному минимуму. Из изображения можно предположить, что в большинстве случаев эта точка будет лежать на одной из вершин ромба, что соответствует решению с одной зануленной компонентой.

Штрафы при малых весах. Предположим, что текущий вектор весов состоит из двух элементов $w = (1, \varepsilon)$, где ε близко к нулю, и мы хотим немного изменить данный вектор по одной из координат. Найдём изменение L_2- и L_1-норм вектора при уменьшении первой компоненты на некоторое положительное число $\delta < \varepsilon$:

$$\|w - (\delta, 0)\|_2^2 = 1 - 2\delta + \delta^2 + \varepsilon^2$$
$$\|w - (\delta, 0)\|_1 = 1 - \delta + \varepsilon$$

Вычислим то же самое для изменения второй компоненты:

$$\|w - (0, \delta)\|_2^2 = 1 - 2\varepsilon\delta + \delta^2 + \varepsilon^2$$
$$\|w - (0, \delta)\|_1 = 1 - \delta + \varepsilon$$

Видно, что с точки зрения L_2-нормы выгоднее уменьшать первую компоненту, а для L_1-нормы оба изменения равноценны. Таким образом, при выборе L_2-регуляризации гораздо меньше шансов, что маленькие веса будут окончательно обнулены.

Проксимальный шаг. *Проксимальные методы* — это класс методов оптимизации, которые хорошо подходят для функционалов с негладкими слагаемыми. Не будем сейчас останавливаться на принципах их работы, а приведём лишь формулу для шага проксимального метода в применении к линейной регрессии с квадратичным функционалом ошибки и L_1-регуляризатором:

$$w^{(k)} = S_{\eta\alpha}\left(w^{(k-1)} - \eta\nabla_w F(w^{(k-1)})\right),$$

где $F(w) = \|Xw - y\|^2$ — функционал ошибки без регуляризатора, η — длина шага, α — коэффициент регуляризации, а функция $S_{\eta\alpha}(w)$ применяется к вектору весов покомпонентно, и для одного элемента выглядит как

$$S_{\eta\alpha}(w_i) = \begin{cases} w_i - \eta\alpha, & w_i > \eta\alpha \\ 0, & |w_i| < \eta\alpha \\ w_i + \eta\alpha, & w_i < -\eta\alpha \end{cases}$$

Из формулы видно, что если на данном шаге значение некоторого веса не очень большое, то на следующем шаге этот вес будет обнулён, причём чем больше коэффициент регуляризации, тем больше весов будут обнуляться.

§2.9 Гиперпараметры

В машинном обучении принято разделять подлежащие настройке величины на *параметры* и *гиперпараметры*. Параметрами называют величины, которые настраиваются по обучающей выборке — например, веса в линейной регрессии. К гиперпараметрам относят величины, которые контролируют сам процесс обучения и не могут быть подобраны по обучающей выборке.

Хорошим примером гиперпараметра является коэффициент регуляризации α. Введение регуляризации мешает модели подгоняться под обучающие данные, и с точки зрения среднеквадратичной ошибки выгодно всегда брать $\alpha = 0$. Разумеется, такой выбор не будет оптимальным с точки зрения качества на новых данных, и поэтому коэффициент регуляризации (как и другие гиперпараметры) следует настраивать по отложенной выборке или с помощью кросс-валидации.

При подборе гиперпараметров по кросс-валидации возникает проблема: мы используем отложенные данные, чтобы выбрать лучший набор гиперпараметров. По сути, отложенная выборка тоже становится обучающей, и показатели качества на ней перестают характеризовать обобщающую способность модели. В таких случаях выборку, на которой настраиваются гиперпараметры, называют валидационной, и при этом выделяют третий, тестовый набор данных, на которых оценивается качество итоговой модели.

§2.10 Преобразование признаков

2.10.1 Нелинейные признаки

С помощью линейной регрессии можно восстанавливать нелинейные зависимости, если провести преобразование признакового пространства:

$$x = (x_1, \ldots, x_d) \to \varphi(x) = (\varphi_1(x), \ldots, \varphi_m(x)).$$

Например, можно перейти к квадратичным признакам:

$$\varphi(x) = (x_1, \ldots, x_d, x_1^2, \ldots, x_d^2, x_1x_2, \ldots, x_{d-1}x_d).$$

Линейная модель над новыми признаками уже сможет приближать любые квадратичные закономерности. Аналогично можно работать и с полиномиальными признаками более высоких порядков.

Возможны и другие преобразования:

- $\log x_j$ — для признаков с тяжёлыми хвостами
- $\exp(\|x - \mu\|^2/\sigma)$ — для измерения близости до некоторой точки
- $\sin(x_j/T)$ — для задач с периодическими зависимостями

2.10.2 Масштабирование

При обучении линейных моделей полезно масштабировать признаки, то есть приводить их к единой шкале. Разберёмся, зачем это нужно.

Рассмотрим функцию $f_1(x) = \frac{1}{2}x_1^2 + \frac{1}{2}x_2^2$, выберем начальное приближение $x^{(0)} = (1, 1)$ и запустим из него градиентный спуск с параметром $\eta = 1$. Окажется, что за один шаг мы сможем сразу попасть в точку минимума.

Теперь «растянем» функцию вдоль одной из осей: $f_2(x) = 50x_1^2 + \frac{1}{2}x_2^2$. При таком же начальном приближении $x^{(0)}$ антиградиент на первой итерации будет равен $(-100, -1)$, и попасть по нему в минимум уже невозможно — более того, при неаккуратном выборе длины шага можно очень далеко уйти от минимума. Пример траектории градиентного спуска при такой форме функции можно найти на рис. 3.

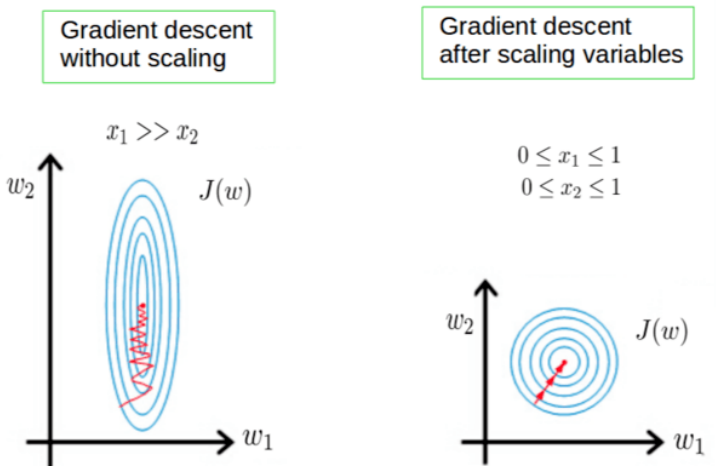

Рис. 3. Траектория градиентного спуска на функционале при признаках разного масштаба.

Аналогичная проблема возникает с функционалом ошибки в линейной регрессии, если один из признаков существенно отличается по масштабу от остальных. Чтобы избежать этого, признаки следует масштабировать — например, путём стандартизации:

$$x_{ij} := \frac{x_{ij} - \mu_j}{\sigma_j},$$

где $\mu_j = \frac{1}{\ell}\sum_{i=1}^{\ell} x_{ij}$, $\sigma_j = \frac{1}{\ell}\sum_{i=1}^{\ell}(x_{ij} - \mu_j)^2$. Или, например, можно масштабировать признаки на отрезок $[0, 1]$:

$$x_{ij} := \frac{x_{ij} - \min_i x_{ij}}{\max_i x_{ij} - \min_i x_{ij}}.$$

3 Линейная классификация

§3.1 Вопросы для самопроверки

1. Как выглядит модель линейной классификации в случае двух классов?
2. Как обучаются линейные классификаторы и для чего нужны верхние оценки пороговой функции потерь? Что такое отступ?
3. Что такое доля правильных ответов? В чём заключаются её проблемы?
4. Что такое матрица ошибок?
5. Что такое точность, полнота?
6. Что такое F-мера? В чем ее перимущество?
7. Что такое средняя точность?
8. Зачем нужно рассматривать Area Under Curve?
9. Что такое AUC-ROC? Опишите алгоритм построения ROC-кривой.
10. Что используется в задачах кредитного скоринга вместо AUC-ROC?
11. В чем проблема AUC-ROC?
12. Что такое AUC-PRC? Зачем это нужно? Опишите алгоритм построения PR-кривой.
13. Запишите функционал логистической регрессии. В чем преимущество обучения на такой функции потерь? Как он связан с методом максимума правдоподобия?
14. Запишите задачу метода опорных векторов для линейно разделимого и неразделимого случаев. Как функционал этой задачи связан с отступом классификатора?
15. Как можно свести задачу многоклассовой классификации к серии задач бинарной классификации?
16. Как бинарную логистическую регрессию напрямую обобщить на случай многих классов?
17. Как бинарный метод опорных векторов обобщить на случай многих классов?
18. Как измеряется качество в задаче многоклассовой классификации? Что такое микро- и макро-усреднение? Какой подход лучше использовать?
19. Как проводится классификация с пересекающимися классами?
20. Как измеряется качество в задаче многоклассовой классификации с пересекающимися классами?

§3.2 Бинарная линейная классификация

Пусть $\mathbb{X} = \mathbb{R}^d$ — пространство объектов, $Y = \{-1, +1\}$ — множество допустимых ответов, $X = \{(x_i, y_i)\}_{i=1}^{\ell}$ — обучающая выборка.

Линейная модель классификации определяется следующим образом:

$$a(x) = \text{sign}\left(\langle w, x\rangle + w_0\right) = \text{sign}\left(\sum_{j=1}^{d} w_j x_j + w_0\right),$$

где $w \in \mathbb{R}^d$ — вектор весов, $w_0 \in \mathbb{R}$ — сдвиг (bias). Заметим, что функция $\text{sign}(z)$ может выдавать ноль при $z = 0$, но в множество ответов ноль не входит. Во-первых, можем понадеяться, что нулевое значение линейной модели — настолько редкое событие, что нам не придётся иметь с этим дело. Во-вторых, можем считать, что если модель выдала ноль, то она не может выбрать класс и отказывается от классификации — что-то вроде выдачи исключения.

Если не сказано иначе, мы будем считать, что среди признаков есть константа, $x_{d+1} = 1$. В этом случае нет необходимости вводить сдвиг w_0, и линейный классификатор можно задавать как

$$a(x) = \text{sign}\langle w, x\rangle.$$

Геометрически линейный классификатор соответствует гиперплоскости с вектором нормали w. Величина скалярного произведения $\langle w, x\rangle$ пропорциональна расстоянию от гиперплоскости до точки x, а его знак показывает, с какой стороны от гиперплоскости находится данная точка. Таким образом, линейный классификатор разделяет пространство на две части с помощью гиперплоскости, и при этом одно полупространство относит к положительному классу, а другое — к отрицательному.

3.2.1 Обучение линейных классификаторов

В задаче регрессии имеется континуум возможных ответов, и при таких условиях достаточно странно требовать полного совпадения ответов модели и истинных ответов — гораздо логичнее говорить об их близости. Более того, как мы выяснили, попытка провести функцию через все обучающие точки легко может привести к переобучению. Способов посчитать близость двух чисел (прогноза и истинного ответа) достаточно много, и поэтому при обсуждении регрессии у нас возникло большое количество функционалов ошибки.

В случае с бинарной классификацией всё гораздо проще: у нас всего два возможных ответа алгоритма и, очевидно, мы хотим видеть как можно больше правильных ответов. Соответствующий функционал называется *долей правильных ответов* (accuracy):

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

Нам будет удобнее решать задачу минимизации, поэтому будем вместо этого использовать долю неправильных ответов:

$$Q(a, X) = \frac{1}{\ell}\sum_{i=1}^{\ell}[a(x_i) \neq y_i] = \frac{1}{\ell}\sum_{i=1}^{\ell}[\text{sign}\langle w, x_i\rangle \neq y_i] \to \min_w \tag{3.1}$$

Этот функционал является дискретным относительно весов, и поэтому искать его минимум с помощью градиентных методов мы не сможем. Более того, у данного функционала может быть много глобальных минимумов — вполне может оказаться, что существует много способов добиться оптимального количества ошибок. Чтобы не пытаться решать все эти проблемы, попытаемся свести задачу к минимизации гладкого функционала.

Отступы. Заметим, что функционал (3.1) можно несколько видоизменить:

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

Здесь возникла очень важная величина $M_i = y_i\langle w, x_i\rangle$, называемая *отступом* (margin). Знак отступа говорит о корректности ответа классификатора (положительный отступ соответствует правильному ответу, отрицательный — неправильному), а его абсолютная величина характеризует степень уверенности классификатора в своём ответе. Напомним, что скалярное произведение $\langle w, x\rangle$ пропорционально расстоянию от разделяющей гиперплоскости до объекта; соответственно, чем ближе отступ к нулю, тем ближе объект к границе классов, тем ниже уверенность в его принадлежности.

Верхние оценки. Функционал (3.1) оценивает ошибку алгоритма на объекте x с помощью пороговой функции потерь $L(M) = [M < 0]$, где аргументом функции является отступ $M = y\langle w, x\rangle$. Оценим эту функцию сверху во всех точках M кроме, может быть, небольшой полуокрестности левее нуля (см. рис. 4):

$$L(M) \leqslant \tilde{L}(M).$$

После этого можно получить верхнюю оценку на функционал (3.1):

$$Q(a, X) \leqslant \frac{1}{\ell}\sum_{i=1}^{\ell}\tilde{L}(y_i\langle w, x_i\rangle) \to \min_w$$

Если верхняя оценка $\tilde{L}(M)$ является гладкой, то и данная верхняя оценка будет гладкой. В этом случае её можно будет минимизировать с помощью, например, градиентного спуска. Если верхнюю оценку удастся приблизить к нулю, то и доля неправильных ответов тоже будет близка к нулю.

Приведём несколько примеров верхних оценок:

1) $\tilde{L}(M) = \log\left(1 + e^{-M}\right)$ — логистическая функция потерь
2) $\tilde{L}(M) = (1 - M)_+ = \max(0, 1 - M)$ — кусочно-линейная функция потерь (используется в методе опорных векторов)
3) $\tilde{L}(M) = (-M)_+ = \max(0, -M)$ — кусочно-линейная функция потерь (соответствует персептрону Розенблатта)
4) $\tilde{L}(M) = e^{-M}$ — экспоненциальная функция потерь
5) $\tilde{L}(M) = 2/(1 + e^{M})$ — сигмоидная функция потерь

Любая из них подойдёт для обучения линейного классификатора.

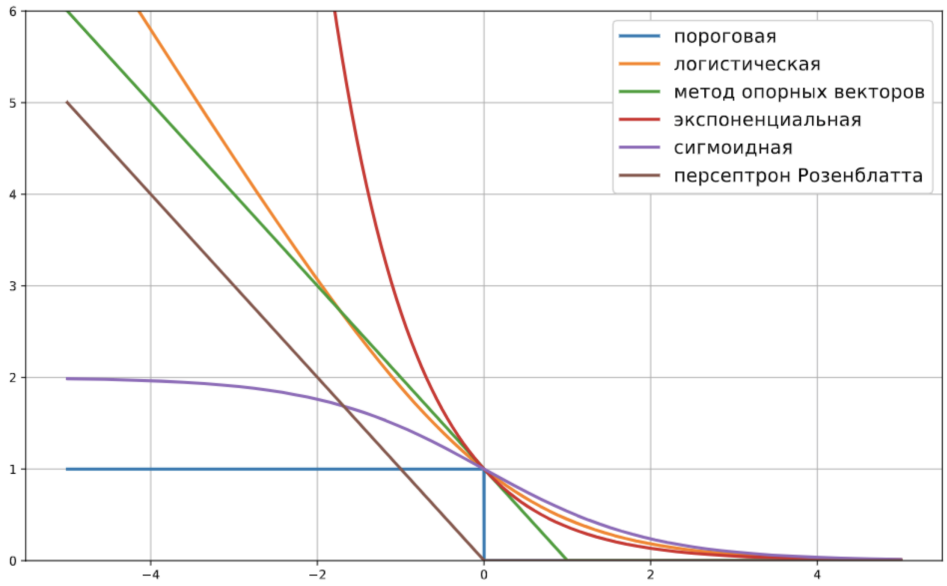

Рис. 4. Верхние оценки на пороговую функцию потерь.

§3.3 Метрики качества классификации

Выше мы разобрали способ сведения задачи обучения линейного классификатора к минимизации гладкого функционала. При этом часто возникает необходимость в изучении различных аспектов качества уже обученного классификатора. Обсудим подробнее распространённые подходы к измерению качества таких моделей.

Будем считать, что классификатор имеет вид $a(x) = \text{sign}(b(x) - t) = 2[b(x) > t] - 1$. Линейная модель имеет именно такую форму, если положить $b(x) = \langle w, x \rangle$ и $t = 0$.

3.3.1 Доля правильных ответов

Наиболее очевидной мерой качества в задаче классификации является доля правильных ответов (accuracy):

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

Данная метрика, однако, имеет существенный недостаток при *несбалансированных классах*. Если взять порог t меньше минимального значения прогноза $b(x)$ на выборке или больше максимального значения, то доля правильных ответов будет равна доле положительных и отрицательных ответов соответственно. Таким образом, если в выборке 950 отрицательных и 50 положительных объектов, то при тривиальном пороге $t = \max_i b(x_i)$ мы получим долю правильных ответов 0.95. Это означает, что доля положительных ответов сама по себе не несет никакой информации о качестве работы алгоритма $a(x)$, и вместе с ней следует анализировать соотношение классов в выборке. Также полезно вместе с долей правильных ответов вычислять *базовую долю* — долю правильных ответов алгоритма, всегда выдающего наиболее мощный класс.

3.3.2 Матрица ошибок

Выше мы убедились, что в случае с *несбалансированными классами* одной доли правильных ответов недостаточно — необходима еще одна метрика качества. В дальнейших вопросах рассмотрим другую, более информативную пару критериев.

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

Таблица 1. Матрица ошибок

Введем сначала понятие матрицы ошибок. Это способ разбить объекты на четыре категории в зависимости от комбинации истинного ответа и ответа алгоритма (см. таблицу 1). Через элементы этой матрицы можно, например, выразить долю правильных ответов:

$$\text{accuracy} = \frac{\text{TP} + \text{TN}}{\text{TP} + \text{FP} + \text{FN} + \text{TN}}.$$

3.3.3 Точность и полнота

Гораздо более информативными критериями являются *точность* (precision) и *полнота* (recall):

$$\text{precision} = \frac{\text{TP}}{\text{TP} + \text{FP}};$$
$$\text{recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

Точность показывает, какая доля объектов, выделенных классификатором как положительные, действительно является положительными. Полнота показывает, какая часть положительных объектов была выделена классификатором.

Рассмотрим, например, задачу предсказания реакции клиента банка на звонок с предложением кредита. Ответ $y = 1$ означает, что клиент возьмет кредит после рекламного звонка, ответ $y = -1$ — что не возьмет. Соответственно, планируется обзванивать только тех клиентов, для которых классификатор $a(x)$ вернет ответ 1. Если классификатор имеет высокую точность, то практически все клиенты, которым будет сделано предложение, откликнутся на него. Если классификатор имеет высокую полноту, то предложение будет сделано практически всем клиентам, которые готовы откликнуться на него. Как правило, можно регулировать точность и полноту, изменяя порог t в классификаторе $a(x) = \text{sign}(b(x) - t) = 2[b(x) > t] - 1$. Если выбрать t большим, то классификатор будет относить к положительному классу небольшое число объектов, что приведет к высокой точности и низкой полноте. По мере уменьшения t точность будет падать, а полнота увеличиваться. Конкретное значение порога выбирается согласно пожеланиям заказчика.

Отметим, что точность и полнота не зависят от соотношения размеров классов. Даже если объектов положительного класса на порядки меньше, чем объектов отрицательного класса, данные показатели будут корректно отражать качество работы алгоритма.

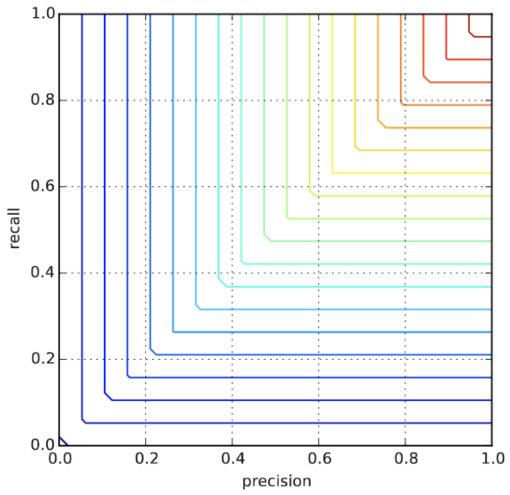

Рис. 5. Линии уровня для минимума из точности и полноты.

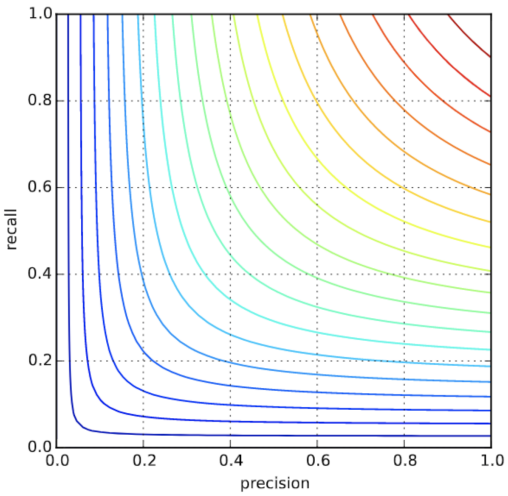

Рис. 6. Линии уровня для F-меры.

3.3.4 F-мера

Существует несколько способов получить один критерий качества на основе точности и полноты. Один из них — F-мера, гармоническое среднее точности и полноты:

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

Среднее гармоническое обладает важным свойством: оно близко к нулю, если хотя бы один из аргументов близок к нулю. Именно поэтому оно является более предпочтительным, чем среднее арифметическое (если алгоритм будет относить все объекты к положительному классу, то он будет иметь recall $= 1$ и precision $\ll 1$, а их среднее арифметическое будет больше $1/2$, что недопустимо). Можно заметить, что F-мера является сглаженной версией минимума из точности и полноты (см. рис. 9 и 10). Отметим, что геометрическое среднее также похоже на сглаженный вариант минимума, но при этом оно менее устойчиво к «выбросам» — например, для точности 0.9 и полноты 0.1 гармоническое среднее будет равно 0.18, а геометрическое 0.3.

3.3.5 Средняя точность

Часто встречаются задачи, в которых целевой признак по-прежнему бинарный, но при этом необходимо ранжировать объекты, а не просто предсказывать их класс. Например, в задаче предсказания реакции клиента можно выдавать сортированный список, чтобы оператор мог в первую очередь позвонить клиентам с наибольшей вероятностью положительного отклика. Поскольку многие алгоритмы возвращают вещественный ответ $b(x)$, который затем бинаризуется по порогу t, то можно просто сортировать объекты по значению $b(x)$. Для измерения качества ранжирования нередко используют *среднюю точность* (average precision, AP):

$$\text{AP} = \frac{1}{\ell_+} \sum_{k=1}^{\ell} [y_{(k)} = 1] \text{precision@}k,$$

где $y_{(k)}$ — ответ k-го по порядку объекта, ℓ_+ — число положительных объектов в выборке, а precision@k — точность среди первых k в списке объектов. Если алгоритм $b(x)$ так ранжирует объекты, что сначала идут все положительные, а затем все отрицательные, то средняя точность будет равна единице; соответственно, чем сильнее положительные документы концентрируются в верхней части списка, тем ближе к единице будет данный показатель.

3.3.6 Про подбор требований метрик качества: связь точности, полноты и доли правильных ответов.

Выше мы отмечали, что высокие значения доли правильных ответов вовсе не влекут за собой высокое качество работы классификатора, и ввели точность и полноту как способ решения этой проблемы. Тем не менее, при выборе точности и полноты в качестве основных метрик, следует соблюдать осторожность при выборе требований к их значениям — как мы увидим из примера, независимость данных метрик от соотношения классов может привести к неочевидным последствиям.

Рассмотрим задачу бинарной классификации с миллионом объектов ($\ell = = 1.000.000$), где доля объектов первого класса составляет 1% ($\ell_+ = 10.000$). Мы знаем, что доля правильных ответов будет вести себя не вполне интуитивно на данной несбалансированной выборке, и поэтому выберем точность и полноту для измерения качества классификаторов. Поскольку мы хотим решить задачу хорошо, то введём требования, что и точность, и полнота должны быть не менее 90%. Эти требования кажутся вполне разумными, если забыть о соотношении классов. Попробуем теперь оценить, какая доля правильных ответов должна быть у классификатора, удовлетворяющего нашим требованиям.

Всего в выборке 10.000 положительных объектов, и для достижения полноты 90% мы должны отнести как минимум 9.000 к положительному классу. Получаем TP = 9000, FN = 1000. Так как точность тоже должна быть не меньше 90%, получаем FP = 1000. Отсюда получаем, что доля правильных ответов должна быть равна $(1 - 2.000/1.000.000) = 99.8\%$! Это крайне высокий показатель, и его редко удаётся достичь на таких выборках во многих предметных областях.

§3.4 Area Under Curve

Выше мы изучили точность, полноту и F-меру, которые характеризуют качество работы алгоритма $a(x) = \text{sign}(b(x) - t)$ при конкретном выборе порога t. Однако зачастую интерес представляет лишь вещественнозначный алгоритм $b(x)$, а порог будет выбираться позже в зависимости от требований к точности и полноте. В таком случае возникает потребность в измерении качества семейства моделей $\{a(x) = \text{sign}(b(x) - - t) \,|\, t \in \mathbb{R}\}$.

Широко используется такая интегральная метрика качества семейства, как *площадь под ROC-кривой* (Area Under ROC Curve, AUC-ROC). Рассмотрим двумерное пространство, одна из координат которого соответствует доле неверно принятых объектов (False Positive Rate, FPR), а другая — доле верно принятых объектов (True

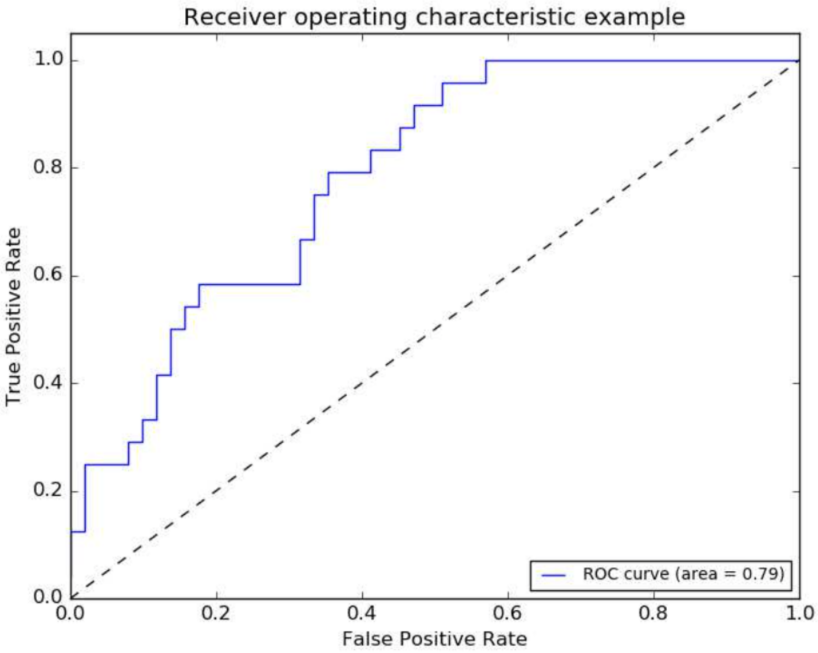

Рис. 7. Пример ROC-кривой.

Positive Rate, TPR):

$$\text{FPR} = \frac{\text{FP}}{\text{FP} + \text{TN}};$$
$$\text{TPR} = \frac{\text{TP}}{\text{TP} + \text{FN}}.$$

Каждый возможный выбор порога t соответствует точке в этом пространстве. Всего различных порогов имеется $\ell + 1$. Максимальный порог $t_{\max} = \max_i b(x_i)$ даст классификатор с TPR $= 0$, FPR $= 0$. Минимальный порог $t_{\min} = \min_i b(x_i) - \varepsilon$ даст TPR $= 1$ и FPR $= 1$. ROC-кривая — это кривая с концами в точках $(0, 0)$ и $(1, 1)$, которая последовательно соединяет точки, соответствующие порогам $b(x_{(1)}) - \varepsilon, b(x_{(1)}), b(x_{(2)}), \ldots, b(x_{(\ell)})$ (см. рис. 11). Площадь под данной кривой называется AUC-ROC, и принимает значения от 0 до 1. Если порог t может быть подобран так, что алгоритм $a(x)$ не будет допускать ошибок, то AUC-ROC будет равен единице; если же $b(x)$ ранжирует объекты случайным образом, то AUC-ROC будет близок к 0.5.

Критерий AUC-ROC имеет большое число интерпретаций — например, он равен вероятности того, что для случайно выбранных положительного объекта x_+ и отрицательного объекта x_- будет выполнено[4] $b(x_+) > b(x_-)$.

3.4.1 Индекс Джини

В задачах кредитного скоринга вместо AUC-ROC часто используется пропорциональная метрика, называемая индексом Джини (Gini index):

$$\text{Gini} = 2\text{AUC} - 1.$$

[4]При условии, что прогнозы на всех объектах выборки попарно различны.

По сути это умноженная на два площадь между ROC-кривой и диагональю, соединяющей точки $(0,0)$ и $(1,1)$.

Отметим, что переход от AUC к индексу Джини приводит к увеличению относительных разниц. Если мы смогли улучшить AUC с 0.8 до 0.9, то это соответствует относительному улучшению в 12.5%. В то же время соответствующие индексы Джини были улучшены с 0.6 до 0.8, то есть на 33.3% — относительное улучшение повысилось почти в три раза!

3.4.2 Чувствительность AUC-ROC к соотношению классов

Рассмотрим задачу выделения математических статей из множества научных статей. Допустим, что всего имеется 1.000.100 статей, из которых лишь 100 относятся к математике. Если нам удастся построить алгоритм $a(x)$, идеально решающий задачу, то его TPR будет равен единице, а FPR — нулю. Рассмотрим теперь плохой алгоритм, дающий положительный ответ на 95 математических и 50.000 нематематических статьях. Такой алгоритм совершенно бесполезен, но при этом имеет $\text{TPR} = 0.95$ и $\text{FPR} = 0.05$, что крайне близко к показателям идеального алгоритма.

Таким образом, если положительный класс существенно меньше по размеру, то AUC-ROC может давать неадекватную оценку качества работы алгоритма, поскольку измеряет долю неверно принятых объектов относительно общего числа отрицательных. Так, алгоритм $b(x)$, помещающий 100 релевантных документов на позиции с 50.001-й по 50.101-ю, будет иметь AUC-ROC 0.95.

3.4.3 AUC-PRC

Избавиться от указанной проблемы с несбалансированными классами можно, перейдя от ROC-кривой к Precision-Recall кривой. Она определяется аналогично ROC-кривой, только по осям откладываются не FPR и TPR, а полнота (по оси абсцисс) и точность (по оси ординат). Критерием качества семейства алгоритмов выступает площадь под PR-кривой (AUC-PR). Данную величину можно аппроксимировать следующим образом. Стартуем из точки $(0,1)$. Будем идти по ранжированной выборке, начиная с первого объекта; пусть текущий объект находится на позиции k. Если он относится к классу «-1», то полнота не меняется, точность падает — соответственно, кривая опускается строго вниз. Если же объект относится к классу «1», то полнота увеличивается на $1/\ell_+$, точность растет, и кривая поднимается вправо и вверх. Площадь под этим участком можно аппроксимировать площадью прямоугольника с высотой, равной $\text{precision@}k$ и шириной $1/\ell_+$. При таком способе подсчета площадь под PR-кривой будет совпадать со средней точностью:

$$\text{AUC-PR} = \frac{1}{\ell_+}\sum_{k=1}^{\ell}[y_k = 1]\text{precision@}k.$$

Отметим, что AUC-PR дает разумные результаты в рассмотренном выше примере с классификацией статей. Так, при размещении 100 релевантных документов на позициях 50.001-50.101 в ранжированном списке, AUC-PR будет равен 0.001.

Несмотря на указанные различия, между ROC- и PR-кривой имеется тесная связь. Так, можно показать, что если ROC-кривая одного алгоритма лежит полно-

стью над ROC-кривой другого алгоритма, то и PR-кривая одного лежит над PR-кривой другого [5].

§3.5 Преамбула к следующим разделам

Ранее мы изучили общий подход к обучению линейных классификаторов, основанный на минимизации верхней оценки:

$$\frac{1}{\ell}\sum_{i=1}^{\ell} L(y_i, \langle w, x_i\rangle) \to \min_w$$

При этом мы привели примеры нескольких верхних оценок $\tilde{L}(M)$, среди которых были

- $L(y,z) = \log(1+\exp(-yz))$ — логистическая функция потерь;
- $L(y,z) = (1-yz)_+$ — кусочно-линейная функция потерь (hinge loss).

Выясним, откуда взялись эти функции потерь, и изучим некоторые их важные свойства.

§3.6 Логистическая регрессия

3.6.1 Оценивание вероятностей

Метод обучения линейного классификатора, который получается при использовании логистической функции потерь, называется логистической регрессией. Основным его свойством является тот факт, что он корректно оценивает вероятность принадлежности объекта к каждому из классов.

Пусть в каждой точке пространства объектов $x \in \mathbb{X}$ задана вероятность $p(y = +1 \,|\, x)$ того, что объект x будет принадлежать классу $+1$. Это означает, что мы допускаем наличие в выборке нескольких объектов с одинаковым признаковым описанием, но с разными значениями целевой переменной; причём если устремить количество объекта x в выборке к бесконечности, то доля положительных объектов среди них будет стремиться к $p(y = +1 \,|\, x)$.

Примером может служить задача предсказания кликов по рекламным баннерам. При посещении одного и того же сайта один и тот же пользователь может как кликнуть, так и не кликнуть по одному и тому же баннеру, из-за чего в выборке могут появиться одинаковые объекты с разными ответами. При этом важно, чтобы классификатор предсказывал именно вероятности классов — если домножить вероятность первого класса на сумму, которую заплатит заказчик в случае клика, то мы получим матожидание прибыли при показе этого баннера. На основе таких матожиданий можно построить алгоритм, выбирающий баннеры для показа пользователю.

Итак, рассмотрим точку x пространства объектов. Как мы договорились, в ней имеется распределение на ответах $p(y = +1 \,|\, x)$. Допустим, алгоритм $b(x)$ возвращает числа из отрезка $[0,1]$. Наша задача — выбрать для него такую процедуру обучения, что в точке x ему будет оптимально выдавать число $p(y = +1 \,|\, x)$. Если в выборке

объект x встречается n раз с ответами $\{y_1, \dots, y_n\}$, то получаем следующее требование:

$$\arg\min_{b\in\mathbb{R}} \frac{1}{n}\sum_{i=1}^{n} L(y_i, b) \approx p(y = +1 \,|\, x).$$

При стремлении n к бесконечности получим, что функционал стремится к матожиданию ошибки:

$$\arg\min_{b\in\mathbb{R}} \mathbb{E}\left[L(y, b) \,|\, x\right] = p(y = +1 \,|\, x).$$

На семинаре будет показано, что этим свойством обладает, например, квадратичная функция потерь $L(y, z) = (y - z)^2$, если в ней для положительных объектов использовать истинную метку $y = 1$, а для отрицательных брать $y = 0$.

Примером функции потерь, которая не позволяет оценивать вероятности, является модуль отклонения $L(y, x) = |y - z|$. Можно показать, что с точки зрения данной функции оптимальным ответом всегда будет либо ноль, либо единица.

Это требование можно воспринимать более просто. Пусть один и тот же объект встречается в выборке 1000 раз, из которых 100 раз он относится к классу $+1$, и 900 раз — к классу -1. Поскольку это один и тот же объект, классификатор должен выдавать один ответ для каждого из тысячи случаев. Можно оценить матожидание функции потерь в данной точке по 1000 примеров при прогнозе b:

$$\mathbb{E}\Big[L(y, b) \,|\, x\Big] \approx \frac{100}{1000} L(1, b) + \frac{900}{1000} L(-1, b).$$

Наше требование, по сути, означает, что оптимальный ответ с точки зрения этой оценки должен быть равен $1/10$:

$$\arg\min_{b\in\mathbb{R}} \left(\frac{100}{1000} L(1, b) + \frac{900}{1000} L(-1, b)\right) = \frac{1}{10}.$$

3.6.2 Правдоподобие и логистические потери

Хотя квадратичная функция потерь и приводит к корректному оцениванию вероятностей, она не очень хорошо подходит для решения задачи классификации. Причиной этому в том числе являются и слишком низкие штрафы за ошибку — так, если объект положительный, а модель выдаёт для него вероятность первого класса $b(x) = 0$, то штраф за это равен всего лишь единице: $(1 - 0)^2 = 1$.

Попробуем сконструировать функцию потерь из других соображений. Если алгоритм $b(x) \in [0, 1]$ действительно выдает вероятности, то они должны согласовываться с выборкой. С точки зрения алгоритма вероятность того, что в выборке встретится объект x_i с классом y_i, равна $b(x_i)^{[y_i=+1]}(1-b(x_i))^{[y_i=-1]}$. Исходя из этого, можно записать правдоподобие выборки (т.е. вероятность получить такую выборку с точки зрения алгоритма):

$$Q(a, X) = \prod_{i=1}^{\ell} b(x_i)^{[y_i=+1]}(1 - b(x_i))^{[y_i=-1]}.$$

Данное правдоподобие можно использовать как функционал для обучения алгоритма — с той лишь оговоркой, что удобнее оптимизировать его логарифм:

$$-\sum_{i=1}^{\ell}\left([y_i=+1]\log b(x_i)+[y_i=-1]\log(1-b(x_i))\right)\to\min$$

Данная функция потерь называется логарифмической (log-loss). Покажем, что она также позволяет корректно предсказывать вероятности. Запишем матожидание функции потерь в точке x:

$$\begin{aligned}\mathbb{E}\Big[L(y,b)\,|\,x\Big] &= \mathbb{E}\Big[-[y=+1]\log b-[y=-1]\log(1-b)\,|\,x\Big]= \\ &= -p(y=+1\,|\,x)\log b-(1-p(y=+1\,|\,x))\log(1-b).\end{aligned}$$

Продифференцируем по b:

$$\frac{\partial}{\partial b}\mathbb{E}\Big[L(y,b)\,|\,x\Big]=-\frac{p(y=+1\,|\,x)}{b}+\frac{1-p(y=+1\,|\,x)}{1-b}=0.$$

Легко видеть, что оптимальный ответ алгоритма равен вероятности положительного класса:

$$b_*=p(y=+1\,|\,x).$$

3.6.3 Логистическая регрессия

Везде выше мы требовали, чтобы алгоритм $b(x)$ возвращал числа из отрезка $[0,1]$. Этого легко достичь, если положить $b(x)=\sigma(\langle w,x\rangle)$, где в качестве σ может выступать любая монотонно неубывающая функция с областью значений $[0,1]$. Мы будем использовать сигмоидную функцию: $\sigma(z)=\frac{1}{1+\exp(-z)}$. Таким образом, чем больше скалярное произведение $\langle w,x\rangle$, тем больше будет предсказанная вероятность. Как при этом можно интерпретировать данное скалярное произведение? Чтобы ответить на этот вопрос, преобразуем уравнение

$$p(y=1\,|\,x)=\frac{1}{1+\exp(-\langle w,x\rangle)}.$$

Выражая из него скалярное произведение, получим

$$\langle w,x\rangle=\log\frac{p(y=+1\,|\,x)}{p(y=-1\,|\,x)}.$$

Получим, что скалярное произведение будет равно логарифму отношения вероятностей классов (log-odds).

Как уже упоминалось выше, при использовании квадратичной функции потерь алгоритм будет пытаться предсказывать вероятности, но данная функция потерь является далеко не самой лучшей, поскольку слабо штрафует за грубые ошибки. Логарифмическая функция потерь подходит гораздо лучше, поскольку не позволяет алгоритму сильно ошибаться в вероятностях.

Подставим трансформированный ответ линейной модели в логарифмическую функцию потерь:

$$-\sum_{i=1}^{\ell}\left([y_i=+1]\log\frac{1}{1+\exp(-\langle w,x_i\rangle)}+[y_i=-1]\log\frac{\exp(-\langle w,x_i\rangle)}{1+\exp(-\langle w,x_i\rangle)}\right)=$$
$$=-\sum_{i=1}^{\ell}\left([y_i=+1]\log\frac{1}{1+\exp(-\langle w,x_i\rangle)}+[y_i=-1]\log\frac{1}{1+\exp(\langle w,x_i\rangle)}\right)=$$
$$=\sum_{i=1}^{\ell}\log\left(1+\exp(-y_i\langle w,x_i\rangle)\right).$$

Полученная функция в точности представляет собой логистические потери, упомянутые в начале. Линейная модель классификации, настроенная путём минимизации данного функционала, называется логистической регрессией. Как видно из приведенных рассуждений, она оптимизирует правдоподобие выборки и дает корректные оценки вероятности принадлежности к положительному классу.

§3.7 Метод опорных векторов

Рассмотрим другой подход к построению функции потерь, основанный на максимизации зазора между классами. Будем рассматривать линейные классификаторы вида

$$a(x)=\operatorname{sign}(\langle w,x\rangle+b),\qquad w\in\mathbb{R}^d, b\in\mathbb{R}.$$

3.7.1 Разделимый случай

Будем считать, что существуют такие параметры w_* и b_*, что соответствующий им классификатор $a(x)$ не допускает ни одной ошибки на обучающей выборке. В этом случае говорят, что выборка *линейно разделима*.

Пусть задан некоторый классификатор $a(x)=\operatorname{sign}(\langle w,x\rangle+b)$. Заметим, что если одновременно умножить параметры w и b на одну и ту же положительную константу, то классификатор не изменится. Распорядимся этой свободой выбора и отнормируем параметры так, что

$$\min_{x\in X}|\langle w,x\rangle+b|=1. \tag{3.2}$$

Можно показать, что расстояние от произвольной точки $x_0\in\mathbb{R}^d$ до гиперплоскости, определяемой данным классификатором, равно

$$\rho(x_0,a)=\frac{|\langle w,x\rangle+b|}{\|w\|}.$$

Тогда расстояние от гиперплоскости до ближайшего объекта обучающей выборки равно

$$\min_{x\in X}\frac{|\langle w,x\rangle+b|}{\|w\|}=\frac{1}{\|w\|}\min_{x\in X}|\langle w,x\rangle+b|=\frac{1}{\|w\|}.$$

Данная величина также называется *отступом (margin)*.

Таким образом, если классификатор без ошибок разделяет обучающую выборку, то ширина его разделяющей полосы равна $\frac{2}{\|w\|}$. Известно, что максимизация ширины разделяющей полосы приводит к повышению обобщающей способности классификатора [6]. Вспомним также, что на повышение обобщающей способности направлена и регуляризация, которая штрафует большую норму весов — а чем больше норма весов, тем меньше ширина разделяющей полосы.

Итак, требуется построить классификатор, идеально разделяющий обучающую выборку, и при этом имеющий максимальный отступ. Запишем соответствующую оптимизационную задачу, которая и будет определять метод опорных векторов для линейно разделимой выборки (hard margin support vector machine):

$$\begin{cases} \frac{1}{2}\|w\|^2 \to \min\limits_{w,b} \\ y_i\left(\langle w, x_i\rangle + b\right) \geqslant 1, \quad i = 1, \ldots, \ell. \end{cases} \tag{3.3}$$

Здесь мы воспользовались тем, что линейный классификатор дает правильный ответ на объекте x_i тогда и только тогда, когда $y_i(\langle w, x_i\rangle + b) \geqslant 0$. Более того, из условия нормировки (3.2) следует, что $y_i(\langle w, x_i\rangle + b) \geqslant 1$.

В данной задаче функционал является строго выпуклым, а ограничения линейными, поэтому сама задача является выпуклой и имеет единственное решение. Более того, задача является квадратичной и может быть решена крайне эффективно.

3.7.2 Неразделимый случай

Рассмотрим теперь общий случай, когда выборку невозможно идеально разделить гиперплоскостью. Это означает, что какие бы w и b мы не взяли, хотя бы одно из ограничений в задаче (3.3) будет нарушено:

$$\exists x_i \in X: \; y_i\left(\langle w, x_i\rangle + b\right) < 1.$$

Сделаем эти ограничения «мягкими», введя штраф $\xi_i \geqslant 0$ за их нарушение:

$$y_i\left(\langle w, x_i\rangle + b\right) \geqslant 1 - \xi_i, \quad i = 1, \ldots, \ell.$$

Отметим, что если отступ объекта лежит между нулем и единицей ($0 \leqslant y_i\left(\langle w, x_i\rangle + b\right) < 1$), то объект верно классифицируется, но имеет ненулевой штраф $\xi > 0$. Таким образом, мы штрафуем объекты за попадание внутрь разделяющей полосы.

Величина $\frac{1}{\|w\|}$ в данном случае называется *мягким отступом (soft margin)*. С одной стороны, мы хотим максимизировать отступ, с другой — минимизировать штраф за неидеальное разделение выборки $\sum_{i=1}^{\ell} \xi_i$. Эти две задачи противоречат друг другу: как правило, излишняя подгонка под выборку приводит к маленькому отступу, и наоборот — максимизация отступа приводит к большой ошибке на обучении. В качестве компромисса будем минимизировать взвешенную сумму двух указанных величин. Приходим к оптимизационной задаче, соответствующей методу опорных векторов для линейно неразделимой выборки (soft margin support vector

machine)

$$\begin{cases} \dfrac{1}{2}\|w\|^2 + C\sum\limits_{i=1}^{\ell}\xi_i \to \min\limits_{w,b,\xi} \\ y_i\left(\langle w, x_i\rangle + b\right) \geqslant 1 - \xi_i, \quad i = 1, \ldots, \ell, \\ \xi_i \geqslant 0, \quad i = 1, \ldots, \ell. \end{cases} \tag{3.4}$$

Чем больше здесь параметр C, тем сильнее мы будем настраиваться на обучающую выборку.

Данная задача также является выпуклой и имеет единственное решение.

3.7.3 Сведение к безусловной задаче

В начале лекции мы планировали разобраться с двумя функциями потерь: логистической и кусочно-линейной. Функционал логистической регрессии действительно имеет вид верхней оценки на долю неправильных ответов, а вот задача метода опорных векторов (3.4) пока выглядит совершенно иначе. Попробуем свести её к задаче безусловной оптимизации.

Перепишем условия задачи:

$$\begin{cases} \xi_i \geqslant 1 - y_i(\langle w, x_i\rangle + b) \\ \xi_i \geqslant 0 \end{cases}$$

Поскольку при этом в функционале требуется, чтобы штрафы ξ_i были как можно меньше, то можно получить следующую явную формулу для них:

$$\xi_i = \max(0, 1 - y_i(\langle w, x_i\rangle + b)).$$

Данное выражение для ξ_i уже учитывает в себе все ограничения задачи (3.4). Значит, если подставить его в функционал, то получим безусловную задачу оптимизации:

$$\frac{1}{2}\|w\|^2 + C\sum_{i=1}^{\ell}\max(0, 1 - y_i(\langle w, x_i\rangle + b)) \to \min_{w,b}$$

Эта задача является негладкой, поэтому решать её может быть достаточно тяжело. Тем не менее, она показывает, что метод опорных векторов, по сути, тоже строит верхнюю оценку вида $L(y, z) = \max(0, 1 - yz)$ на долю ошибок, и добавляет к ней стандартную квадратичную регуляризацию.

§3.8 Многоклассовая линейная классификация

Мы разобрались с общим подходом к решению задачи бинарной классификации, а также изучили свойства двух конкретных методов: логистической регрессии и метода опорных векторов. Теперь мы перейдём к более общей, многоклассовой постановке задачи классификации, и попытаемся понять, как можно свести её к уже известным нам методам.

Будем считать, что каждый объект относится к одному из K классов: $\mathbb{Y} = \{1, \ldots, K\}$.

3.8.1 Сведение к серии бинарных задач

Вполне естественно попытаться свести многоклассовую задачу к набору бинарных. Существует достаточно много способов сделать это — мы перечислим лишь два самых популярных, а об остальных можно почитать, например, [7, раздел 4.2.7]

Один против всех (one-versus-all). Обучим K линейных классификаторов $b_1(x), \ldots, b_K(x)$, выдающих оценки принадлежности классам $1, \ldots, K$ соответственно. Например, в случае с линейными моделями эти модели будут иметь вид

$$b_k(x) = \langle w_k, x \rangle + w_{0k}.$$

Классификатор с номером k будем обучать по выборке $(x_i, 2[y_i = k] - 1)_{i=1}^{\ell}$; иными словами, мы учим классификатор отличать k-й класс от всех остальных.

Итоговый классификатор будет выдавать класс, соответствующий самому уверенному из бинарных алгоритмов:

$$a(x) = \arg\max_{k \in \{1,\ldots,K\}} b_k(x).$$

Проблема данного подхода заключается в том, что каждый из классификаторов $b_1(x), \ldots, b_K(x)$ обучается на своей выборке, и выходы этих классификаторов могут иметь разные масштабы, из-за чего сравнивать их будет неправильно [8]. Нормировать вектора весов, чтобы они выдавали ответы в одной и той же шкале, не всегда может быть разумным решением — так, в случае с SVM веса перестанут являться решением задачи, поскольку нормировка изменит норму весов.

Все против всех (all-versus-all). Обучим C_K^2 классификаторов $a_{ij}(x)$, $i, j = 1, \ldots, K$, $i \neq j$. Например, в случае с линейными моделями эти модели будут иметь вид

$$b_k(x) = \text{sign}(\langle w_k, x \rangle + w_{0k}).$$

Классификатор $a_{ij}(x)$ будем настраивать по подвыборке $X_{ij} \subset X$, содержащей только объекты классов i и j:

$$X_{ij} = \{(x_n, y_n) \in X \mid [y_n = i] = 1 \text{ или } [y_n = j] = 1\}.$$

Соответственно, классификатор $a_{ij}(x)$ будет выдавать для любого объекта либо класс i, либо класс j.

Чтобы классифицировать новый объект, подадим его на вход каждого из построенных бинарных классификаторов. Каждый из них проголосует за своей класс; в качестве ответа выберем тот класс, за который наберется больше всего голосов:

$$a(x) = \arg\max_{k \in \{1,\ldots,K\}} \sum_{i=1}^{K} \sum_{j \neq i} [a_{ij}(x) = k].$$

3.8.2 Многоклассовая логистическая регрессия

Некоторые методы бинарной классификации можно напрямую обобщить на случай многих классов. Выясним, как это можно проделать с логистической регрессией.

В логистической регрессии для двух классов мы строили линейную модель $b(x) = \langle w, x\rangle + w_0$, а затем переводили её прогноз в вероятность с помощью сигмоидной функции $\sigma(z) = \frac{1}{1+\exp(-z)}$. Допустим, что мы теперь решаем многоклассовую задачу и построили K линейных моделей $b_k(x) = \langle w_k, x\rangle + w_{0k}$, каждая из которых даёт оценку принадлежности объекта одному из классов. Как преобразовать вектор оценок $(b_1(x), \ldots, b_K(x))$ в вероятности? Для этого можно воспользоваться оператором $\text{SoftMax}(z_1, \ldots, z_K)$, который производит «нормировку» вектора:

$$\text{SoftMax}(z_1, \ldots, z_K) = \left(\frac{\exp(z_1)}{\sum_{k=1}^K \exp(z_k)}, \ldots, \frac{\exp(z_K)}{\sum_{k=1}^K \exp(z_k)}\right).$$

В этом случае вероятность k-го класса будет выражаться как

$$P(y = k \,|\, x, w) = \frac{\exp\left(\langle w_k, x\rangle + w_{0k}\right)}{\sum_{j=1}^K \exp\left(\langle w_j, x\rangle + w_{0j}\right)}.$$

Обучать эти веса предлагается с помощью метода максимального правдоподобия — так же, как и в случае с двухклассовой логистической регрессией:

$$\sum_{i=1}^{\ell} \log P(y = y_i \,|\, x_i, w) \to \max_{w_1,\ldots,w_K}.$$

3.8.3 Многоклассовый метод опорных векторов

В алгоритме «один против всех» мы *независимо* строили свой классификатор за каждый класс. Попробуем теперь строить эти классификаторы одновременно, в рамках одной оптимизационной задачи. Подходов к обобщению метода опорных векторов на многоклассовый случай достаточно много; мы разберём способ, описанный в работе [9].

Для простоты будем считать, что в выборке имеется константный признак, и не будет явно указывать сдвиг b. Будем настраивать K наборов параметров $w_1, \ldots, w_K$, и итоговый алгоритм определим как

$$a(x) = \arg\max_{k\in\{1,\ldots,K\}} \langle w_k, x\rangle.$$

Рассмотрим следующую функцию потерь:

$$\max_k \Big\{\langle w_k, x\rangle + 1 - [k = y(x)]\Big\} - \langle w_{y(x)}, x\rangle. \tag{3.5}$$

Разберемся сначала с выражением, по которому берется максимум. Если $k = y(x)$, то оно равно $\langle w_k, x\rangle$; в противном же случае оно равно $\langle w_k, x\rangle + 1$. Если оценка за верный класс больше оценок за остальные классы хотя бы на единицу, то максимум будет достигаться на $k = y(x)$; в этом случае потеря будет равна нулю. Иначе же потеря будет больше нуля. Здесь можно увидеть некоторую аналогию с

бинарным SVM: мы штрафуем не только за неверный ответ на объекте, но и за неуверенную классификацию (за попадание объекта в разделяющую полосу).

Рассмотрим сначала линейно разделимую выборку — т.е. такую, что существуют веса $w_{1*}, \ldots, w_{K*}$, при которых потеря (3.5) равна нулю. В бинарном SVM мы строили классификатор с максимальным отступом. Известно, что аналогом отступа для многоклассового случая является норма Фробениуса матрицы W, k-я строка которой совпадает с w_k:

$$\rho = \frac{1}{\|W\|^2} = \frac{1}{\sum_{k=1}^{K}\sum_{j=1}^{d} w_{kj}^2}.$$

Получаем следующую задачу:

$$\begin{cases} \dfrac{1}{2}\|W\|^2 \to \min\limits_{W} \\ \langle w_{y_i}, x_i\rangle + [y_i = k] - \langle w_k, x_i\rangle \geqslant 1, \quad i = 1, \ldots, \ell; k = 1, \ldots, K. \end{cases} \tag{3.6}$$

Перейдем теперь к общему случаю. Как и в бинарном методе опорных векторов, перейдем к мягкой функции потерь, введя штрафы за неверную или неуверенную классификацию. Получим задачу

$$\begin{cases} \dfrac{1}{2}\|W\|^2 + C\sum\limits_{i=1}^{\ell}\xi_i \to \min\limits_{W,\xi} \\ \langle w_{y_i}, x_i\rangle + [y_i = k] - \langle w_k, x_i\rangle \geqslant 1 - \xi_i, \quad i = 1, \ldots, \ell; k = 1, \ldots, K; \\ \xi_i \geqslant 0, \quad i = 1, \ldots, \ell. \end{cases} \tag{3.7}$$

Решать задачу (3.7) можно, например, при помощи пакета $\text{SVM}^{\text{multiclass}}$.

Отметим, что такой подход решает проблему с несоизмеримостью величин, выдаваемых отдельными классификаторами (о которой шла речь в подходе «один против всех»): классификаторы настраиваются одновременно, и выдаваемые ими оценки должны правильно соотноситься друг с другом, чтобы удовлетворять ограничениям.

§3.9 Метрики качества многоклассовой классификации

В многоклассовых задачах, как правило, стараются свести подсчет качества к вычислению одной из рассмотренных выше двухклассовых метрик. Выделяют два подхода к такому сведению: микро- и макро-усреднение.

Пусть выборка состоит из K классов. Рассмотрим K двухклассовых задач, каждая из которых заключается в отделении своего класса от остальных, то есть целевые значения для k-й задаче вычисляются как $y_i^k = [y_i = k]$. Для каждой из них можно вычислить различные характеристики (TP, FP, и т.д.) алгоритма $a^k(x) = [a(x) = k]$; будем обозначать эти величины как $\text{TP}_k, \text{FP}_k, \text{FN}_k, \text{TN}_k$. Заметим, что в двухклассовом случае все метрики качества, которые мы изучали, выражались через эти элементы матрицы ошибок.

При микро-усреднении сначала эти характеристики усредняются по всем классам, а затем вычисляется итоговая двухклассовая метрика — например, точность, полнота или F-мера. Например, точность будет вычисляться по формуле

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

где, например, $\overline{\text{TP}}$ вычисляется по формуле

$$\overline{\text{TP}} = \frac{1}{K}\sum_{k=1}^{K}\text{TP}_k.$$

При макро-усреднении сначала вычисляется итоговая метрика для каждого класса, а затем результаты усредняются по всем классам. Например, точность будет вычислена как

$$\text{precision}(a, X) = \frac{1}{K}\sum_{k=1}^{K}\text{precision}_k(a, X); \qquad \text{precision}_k(a, X) = \frac{\text{TP}_k}{\text{TP}_k + \text{FP}_k}.$$

Если какой-то класс имеет очень маленькую мощность, то при микро-усреднении он практически никак не будет влиять на результат, поскольку его вклад в средние TP, FP, FN и TN будет незначителен. В случае же с макро-вариантом усреднение проводится для величин, которые уже не чувствительны к соотношению размеров классов (если мы используем, например, точность или полноту), и поэтому каждый класс внесет равный вклад в итоговую метрику.

§3.10 Многоклассовая классификация с пересекающимися классами

Усложним постановку задачи. Будем считать, что в задаче K классов, но теперь они могут *пересекаться* — каждый объект может относиться одновременно к нескольким классам. Это означает, что каждому объекту x соответствует вектор $y \in \{0, 1\}^K$, показывающий, к каким классам данный объект относится. Соответственно, обучающей выборке X будет соответствовать матрица $Y \in \{0, 1\}^{\ell \times K}$, описывающая метки объектов; её элемент y_{ik} показывает, относится ли объект x_i к классу k. Данная задача в англоязычной литературе носит название *multi-label classification*. К ней может относиться, например, определение тэгов для фильма или категорий для статьи на Википедии.

3.10.1 Независимая классификация (Binary relevance)

Самый простой подход к решению данной задачи — предположить, что все классы независимы, и определять принадлежность объекта к каждому отдельным классификатором. Это означает, что мы обучаем K бинарных классификаторов $a_1(x), \ldots, a_K(x)$, причём классификатор $b_k(x)$ обучается по выборке $(x_i, y_{ik})_{i=1}^{\ell}$. Для нового объекта x целевая переменная оценивается как $(a_1(x), \ldots, a_K(x))$.

Основная проблема данного подхода состоит в том, что никак не учитываются возможные связи между отдельными классами. Тем не менее, такие связи могут иметь место — например, категории на Википедии имеют древовидную структуру, и если мы с большой уверенностью отнесли статью к некоторой категории, то из этого может следовать, что статья относится к одной из категорий-потомков.

3.10.2 Стекинг классификаторов

Для учёта корреляций между классами можно воспользоваться следующим несложным подходом. Разобьём обучающую выборку X на две части X_1 и X_2. На первой

части обучим K независимых классификаторов $b_1(x), \ldots, b_K(x)$. Далее сформируем для каждого объекта $x_i \in X_2$ из второй выборки признаковое описание, состоящее из прогнозов наших классификаторов:

$$x'_{ik} = b_k(x_i), \quad x_i \in X_2,$$

получив тем самым выборку X'_2. Обучим на ней новый набор классификаторов $a_1(x'), \ldots, a_K(x')$, каждый из которых определяет принадлежность объекта к одному из классов. При этом все новые классификаторы опираются на прогнозы классификаторов первого этапа $b_1(x), \ldots, b_K(x)$, и поэтому могут обнаружить связи между различными классами. Такой подход называется *стекингом* и достаточно часто используется в машинном обучении для усиления моделей.

Отметим, что обучать классификаторы $b_k(x)$ и $a_k(x)$ на одной и той же выборке было бы плохой идеей. Прогнозы базовых моделей $b_k(x)$ содержат в себе информацию об обучающей выборке X_1; получается, что новые признаки $x'_{ik} = b_k(x_i)$, посчитанные по этой же выборке, по сути будут «подглядывать» в целевую переменную, и обучение на них новой модели просто приведёт к переобучению.

Посмотрим на эту проблему несколько иначе. Допустим, мы обучили на выборке X_1 алгоритм $b(x)$, а затем на этой же выборке обучили второй алгоритм $a(b(x))$, использующий в качестве единственного признака результат работы $b(x)$. Если модель $b(x)$ не переобучилась и будет показывать на новых данных такое же качество, как и на обучающей выборк, то никаких проблем не будет. Тем не менее, обычно модели хотя бы немного переобучаются. Будем считать, что на обучении $b(x)$ имеет среднее отклонение от целевой переменной в 5%, а на новых данных она в среднем ошибается на 10% из-за переобучения. Тогда модель $a(x)$ будет рассчитывать на среднее отклонение в 5%, но на новых данных ситуация будет другой — фактически, изменится распределение её признака, что приведёт к не самым лучшим последствиям.

§3.11 Метрики качества многоклассовой классификации с пересекающимися классами

Обозначим через Y_i множество классов, которым объект x_i принадлежит на самом деле, а через Z_i — множество классов, к которым объект был отнесён алгоритмом $a(x)$.

Вполне логичной мерой ошибки будет хэммингово расстояние между этими множествами — то есть доля классов, факт принадлежности которым угадан неверно:

$$\text{hamming}(a, X) = \frac{1}{\ell} \sum_{i=1}^{\ell} \frac{|Y_i \setminus Z_i| + |Z_i \setminus Y_i|}{K}.$$

Данную метрику необходимо минимизировать.

Стандартные метрики качества классификации можно обобщить на multilabel-задачу так же, как и на случай с непересекающимися классами — через микро- или макро-усреднение. Есть и несколько другой подход к обобщению основных метрик

качества:

$$\text{accuracy}(a, X) = \frac{1}{\ell}\sum_{i=1}^{\ell}\frac{|Y_i \cap Z_i|}{|Y_i \cup Z_i|},$$

$$\text{precision}(a, X) = \frac{1}{\ell}\sum_{i=1}^{\ell}\frac{|Y_i \cap Z_i|}{|Z_i|},$$

$$\text{recall}(a, X) = \frac{1}{\ell}\sum_{i=1}^{\ell}\frac{|Y_i \cap Z_i|}{|Y_i|}.$$

Все эти метрики необходимо максимизировать.

4 Категориальные признаки

Обсудим методы работы с категориальными признаками и поймём, почему на таких данных чаще всего используются линейные модели.

Допустим, в выборке имеется категориальный признак, значение которого на объекте x будем обозначать через $f(x)$. Будем считать, что он принимает значения из множества $U = \{u_1, \ldots, u_n\}$. Чтобы использовать такой признак в линейных моделях, необходимо сначала его закодировать. Существует много подходов к использованию категориальных признаков — о многих из них можно узнать в работе [11], мы же рассмотрим несколько наиболее популярных.

§4.1 Вопросы для самопроверки

1. Как можно использовать категориальные признаки в линейных моделях?
2. Почему использование счётчиков может привести к переобучению? Какие методы борьбы с этой проблемой счётчиков вам известны?

§4.2 Бинарное кодирование (one-hot encoding)

Простейший способ — создать n индикаторов, каждый из которых будет отвечать за одно из возможных значений признака. Иными словами, мы формируем n бинарных признаков $g_1(x), \ldots, g_n(x)$, которые определяются как

$$g_j(x) = [f(x) = u_j].$$

Главная проблема этого подхода заключается в том, что на выборках, где категориальные признаки имеют миллионы возможных значений, мы получим огромное количество признаков. Линейные модели хорошо справляются с такими ситуациями за счёт небольшого количества параметров и достаточно простых методов обучения, и поэтому их часто используют на выборках с категориальными признаками.

§4.3 Бинарное кодирование с хэшированием

Рассмотрим модификацию бинарного кодирования, которая позволяет ускорить процесс вычисления признаков. Выберем хэш-функцию $h : U \to \{1, 2, \ldots, B\}$, которая переводит значения категориального признака в числа от 1 до B. После этого бинарные признаки можно индексировать значениями хэш-функции:

$$g_j(x) = [h(f(x)) = j], \quad j = 1, \ldots, B.$$

Основное преимущество этого подхода состоит в том, что отпадает необходимость в хранении соответствий между значениями категориального признака и индексами бинарных признаков. Теперь достаточно лишь уметь вычислять саму хэш-функцию, которая уже автоматически даёт правильную индексацию.

Также хэширование позволяет понизить количество признаков (если $B < |U|$), причём, как правило, это не приводит к существенной потере качества. Это можно объяснить и с помощью интуиции — если у категориального признака много значений, то, скорее всего, большая часть этих значений крайне редко встречается в

выборке, и поэтому не несёт в себе много информации; основную ценность представляют значения $u \in U$, которые много раз встречаются в выборке, поскольку для них можно установить связи с целевой переменной. Хэширование, по сути, случайно группирует значения признака — в одну группу попадают значения, получающие одинаковые индексы $h(u)$. Поскольку «частых» значений не так много, вероятность их попадания в одну группу будет ниже, чем вероятность группировки редких значений.

§4.4 Счётчики

Попытаемся закодировать признаки более экономно. Заметим, что значения категориального признака нужны нам не сами по себе, а лишь для предсказания класса. Соответственно, если два возможных значения u_i и u_j характерны для одного и того же класса, то можно их и не различать.

Определим наш способ кодирования. Вычислим для каждого значения u категориального признака $(K+1)$ величин:

$$\text{counts}(u, X) = \sum_{(x,y)\in X} [f(x) = u],$$
$$\text{successes}_k(u, X) = \sum_{(x,y)\in X} [f(x) = u][y = k], \quad k = 1, \ldots, K.$$

По сути, мы посчитали количество объектов с данным значением признака, а также количество объектов различных классов среди них.

После того, как данные величины подсчитаны, заменим наш категориальный признак $f(x)$ на K вещественных $g_1(x), \ldots, g_K(x)$:

$$g_k(x, X) = \frac{\text{successes}_k(f(x), X) + c_k}{\text{counts}(f(x), X) + \sum_{m=1}^{K} c_m}, \quad k = 1, \ldots, K.$$

Здесь признак $g_k(x)$ фактически оценивает вероятность $p(y = k \mid f(x))$. Величины c_k являются своего рода регуляризаторами и предотвращают деление на ноль в случае, если в выборке X нет ни одного объекта с таким значением признака. Для простоты можно полагать их все равными единице: $c_1 = \cdots = c_K = 1$. У признаков $g_k(x)$ есть много названий: счётчики [5], правдоподобия и т.д.

Отметим, что $g_k(x)$ можно воспринимать как простейший классификатор — значит, при обучении полноценного классификатора на признаках-счётчиках мы рискуем столкнуться с переобучением из-за «утечки» целевой переменной в значения признаков (мы уже обсуждали эту проблему, разбираясь со стекингом). Чтобы избежать переобучения, как правило, пользуются подходом, аналогичным кроссвалидации. Выборка разбивается на m частей $X_1, \ldots, X_m$, и для подвыборки X_i значения признаков вычисляются на основе статистик, подсчитанных по всем остальным частям:

$$x \in X_i \Rightarrow g_k(x) = g_k(x, X \setminus X_i).$$

[5] http://blogs.technet.com/b/machinelearning/archive/2015/02/17/big-learning-made-easy-with-counts.aspx

Можно взять число блоков разбиения равным числу объектов $m = \ell$ — в этом случае значения признаков для каждого объекта будут вычисляться по статистикам, подсчитанным по всем остальным объектам.

Для тестовой выборки значения целевой переменной неизвестны, поэтому на таких объектах признаки-счётчики вычисляются на основе статистик $\text{successes}(u, X)$ и $\text{counts}(u, X)$, подсчитанных по всей обучающей выборке.

При использовании счётчиков нередко используют следующие трюки:

1) К признакам можно добавлять не только дроби $g_k(x)$, но и значения $\text{counts}(f(x), X)$ и $\text{successes}_k(f(x), X)$.

2) Можно сгенерировать парные категориальные признаки, т.е. для каждой пары категориальных признаков $f_i(x)$ и $f_j(x)$ создать новый признак $f_{ij}(x) = = (f_i(x), f_j(x))$. После этого счётчики можно вычислить и для парных признаков; при этом общее количество признаков существенно увеличится, но при этом, как правило, прирост качества тоже оказывается существенным.

3) Если у категориальных признаков много возможных значений, то хранение статистик $\text{counts}(u, X)$ и $\text{successes}_k(u, X)$ может потребовать существенного количества памяти. Для экономии памяти можно хранить статистику не по самим значениям категориального признака $u \in U$, а по хэшам от этих значений $h(u)$. Регулируя количество возможных значений хэш-функции, можно ограничивать количество используемой памяти.

4) Можно вычислять несколько счётчиков для разных значений параметров $c_1, \ldots, c_K$.

5) Можно все редкие значения категориального признака объединить в одно, поскольку скорее всего, для редких значений не получится качественно оценить статистики successes_k и counts. Благодаря этому можно будет сократить расходы на память при хранении статистики. Более того, можно предположить, что все редкие значения похожи, и относить к данной «объединённой» группе и новые значения признака, которые впервые встретятся на тестовой выборке.

Отметим, что данный подход работает только для задач классификации. В то же время можно пытаться адаптировать его и для задач регрессии, вычисляя несколько бинаризаций целевой переменной по разным порогам, и для каждой такой бинаризации вычисляя счётчики.

5 Решающие деревья

Мы уделили достаточно много внимания линейным методам, которые обладают рядом важных достоинств: быстро обучаются, способны работать с большим количеством объектов и признаков, имеют небольшое количество параметров, легко регуляризуются. При этом у них есть и серьёзный недостаток — они могут восстанавливать только линейные зависимости между целевой переменной и признаками. Конечно, можно добавлять в выборку новые признаки, которые нелинейно зависят от исходных, но этот подход является чисто эвристическим, требует выбора типа нелинейности, а также всё равно ограничивает сложность модели сложностью признаков (например, если признаки квадратичные, то и модель сможет восстанавливать только зависимости второго порядка).

Разберёмся с решающими деревьями — семейством моделей, которые позволяют восстанавливать нелинейные зависимости произвольной сложности.

Решающие деревья хорошо описывают процесс принятия решения во многих ситуациях. Например, когда клиент приходит в банк и просит выдать ему кредит, то сотрудник банка начинает проверять условия:

1) Какой возраст у клиента? Если меньше 18, то отказываем в кредите, иначе продолжаем.

2) Какая зарплата у клиента? Если меньше 50 тысяч рублей, то переходим к шагу 3, иначе к шагу 4.

3) Какой стаж у клиента? Если меньше 10 лет, то не выдаем кредит, иначе выдаем.

4) Есть ли у клиента другие кредиты? Если есть, то отказываем, иначе выдаем.

Такой алгоритм, как и многие другие, очень хорошо описывается решающим деревом. Это отчасти объясняет их популярность.

Первые работы по использованию решающих деревьев для анализа данных появились в 60-х годах, и с тех пор несколько десятилетий им уделялось очень большое внимание. Несмотря на свою интерпретируемость и высокую выразительную способность, деревья крайне трудны для оптимизации из-за свой дискретной структуры — дерево нельзя продифференцировать по параметрам и найти с помощью градиентного спуска хотя бы локальный оптимум. Более того, даже число параметров у них не является постоянным и может меняться в зависимости от глубины, выбора критериев дробления и прочих деталей. Из-за этого все методы построения решающих деревьев являются жадными и эвристичными.

На сегодняшний день решающие деревья не очень часто используются как отдельные методы классификации или регрессии. В то же время, как оказалось, они очень хорошо объединяются в композиции — решающие леса, которые являются одними из наиболее сильных и универсальных моделей.

§5.1 Вопросы для самопроверки

1. Что такое решающее дерево?

2. Опишите жадный алгоритм обучения решающего дерева. Как решающее дерево строит прогноз для объекта?

3. Как обучаются решающие деревья в задачах классификации и регрессии (критерии информативности)?
4. Какие критерии останова вы знаете?
5. Как учитывать категориальные признаки в решающих деревьях?
6. Как линейные модели и решающие деревья связаны с друг другом?
7. Какие гиперпараметры имеются у решающих деревьев?
8. Почему с помощью бинарного решающего дерева можно достичь нулевой ошибки на обучающей выборке без повторяющихся объектов?

§5.2 Определение решающего дерева

Рассмотрим бинарное дерево, в котором:

- каждой внутренней вершине v приписана функция (или предикат) $\beta_v : \mathbb{X} \to \{0, 1\}$;
- каждой листовой вершине v приписан прогноз $c_v \in Y$ (в случае с классификацией листу также может быть приписан вектор вероятностей).

Рассмотрим теперь алгоритм $a(x)$, который стартует из корневой вершины v_0 и вычисляет значение функции β_{v_0}. Если оно равно нулю, то алгоритм переходит в левую дочернюю вершину, иначе в правую, вычисляет значение предиката в новой вершине и делает переход или влево, или вправо. Процесс продолжается, пока не будет достигнута листовая вершина; алгоритм возвращает тот класс, который приписан этой вершине. Такой алгоритм называется *бинарным решающим деревом*.

На практике в большинстве случаев используются одномерные предикаты β_v, которые сравнивают значение одного из признаков с порогом:

$$\beta_v(x; j, t) = [x_j < t].$$

Существуют и многомерные предикаты, например:

- линейные $\beta_v(x) = [\langle w, x\rangle < t]$;
- метрические $\beta_v(x) = [\rho(x, x_v) < t]$, где точка x_v является одним из объектов выборки любой точкой признакового пространства.

Многомерные предикаты позволяют строить ещё более сложные разделяющие поверхности, но очень редко используются на практике — например, из-за того, что усиливают и без того выдающиеся способности деревьев к переобучению. Далее мы будем говорить только об одномерных предикатах.

§5.3 Построение деревьев

Легко убедиться, что для любой выборки можно построить решающее дерево, не допускающее на ней ни одной ошибки — даже с простыми одномерными предикатами можно сформировать дерево, в каждом листе которого находится ровно по одному объекту выборки. Скорее всего, это дерево будет переобученным и не сможет показать хорошее качество на новых данных. Можно было бы поставить задачу поиска дерева, которое является минимальным (с точки зрения количества листьев) среди всех деревьев, не допускающих ошибок на обучении — в этом случае можно было бы надеяться на наличие у дерева обобщающей способности. К сожалению, эта задача является NP-полной, и поэтому приходится ограничиваться жадными алгоритмами построения дерева.

Опишем базовый жадный алгоритм построения бинарного решающего дерева. Начнем со всей обучающей выборки X и найдем наилучшее ее разбиение на две части $R_1(j,t) = \{x \,|\, x_j < t\}$ и $R_2(j,t) = \{x \,|\, x_j \geqslant t\}$ с точки зрения заранее заданного функционала качества $Q(X,j,t)$. Найдя наилучшие значения j и t, создадим корневую вершину дерева, поставив ей в соответствие предикат $[x_j < t]$. Объекты разобьются на две части — одни попадут в левое поддерево, другие в правое. Для каждой из этих подвыборок рекурсивно повторим процедуру, построив дочерние вершины для корневой, и так далее. В каждой вершине мы проверяем, не выполнилось ли некоторое условие останова — и если выполнилось, то прекращаем рекурсию и объявляем эту вершину листом. Когда дерево построено, каждому листу ставится в соответствие ответ. В случае с классификацией это может быть класс, к которому относится больше всего объектов в листе, или вектор вероятностей (скажем, вероятность класса может быть равна доле его объектов в листе). Для регрессии это может быть среднее значение, медиана или другая функция от целевых переменных объектов в листе. Выбор конкретной функции зависит от функционала качества в исходной задаче.

Решающие деревья могут обрабатывать пропущенные значения — ситуации, в которых для некоторых объектов неизвестны значения одного или нескольких признаков. Для этого необходимо модифицировать процедуру разбиения выборки в вершине, что можно сделать несколькими способами.

После того, как дерево построено, можно провести его *стрижку* (pruning) — удаление некоторых вершин с целью понижения сложности и повышения обобщающей способности. Существует несколько подходов к стрижке, о которых мы немного упомянем ниже.

Таким образом, конкретный метод построения решающего дерева определяется:

1) Видом предикатов в вершинах;
2) Функционалом качества $Q(X,j,t)$;
3) Критерием останова;
4) Методом обработки пропущенных значений;
5) Методом стрижки.

Также могут иметь место различные расширения, связанные с учетом весов объектов, работой с категориальными признакам и т.д. Ниже мы обсудим варианты каждого из перечисленных пунктов.

§5.4 Критерии информативности

При построении дерева необходимо задать *функционал качества*, на основе которого осуществляется разбиение выборки на каждом шаге. Обозначим через R_m множество объектов, попавших в вершину, разбиваемую на данном шаге, а через R_ℓ и R_r — объекты, попадающие в левое и правое поддерево соответственно при заданном предикате. Мы будем использовать функционалы следующего вида:

$$Q(R_m, j, s) = H(R_m) - \frac{|R_\ell|}{|R_m|} H(R_\ell) - \frac{|R_r|}{|R_m|} H(R_r).$$

Здесь $H(R)$ — это *критерий информативности* (impurity criterion), который оценивает качество распределения целевой переменной среди объектов множества R. Чем меньше разнообразие целевой переменной, тем меньше должно быть значение критерия информативности — и, соответственно, мы будем пытаться минимизировать его значение. Функционал качества $Q(R_m, j, s)$ мы при этом будем максимизировать.

Как уже обсуждалось выше, в каждом листе дерево будет выдавать константу — вещественное число, вероятность или класс. Исходя из этого, можно предложить оценивать качество множества объектов R тем, насколько хорошо их целевые переменные предсказываются константой (при оптимальном выборе этой константы):

$$H(R) = \min_{c \in \mathbb{Y}} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} L(y_i, c),$$

где $L(y, c)$ — некоторая функция потерь. Теперь обсудим, какие именно критерии информативности часто используют в задачах регрессии и классификации.

5.4.1 Регрессия

Как обычно, в регрессии выберем квадрат отклонения в качестве функции потерь. В этом случае критерий информативности будет выглядеть как

$$H(R) = \min_{c \in \mathbb{Y}} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} (y_i - c)^2.$$

Как известно, минимум в этом выражении будет достигаться на среднем значении целевой переменной. Значит, критерий можно переписать в следующем виде:

$$H(R) = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} \left(y_i - \frac{1}{|R|} \sum_{(x_j, y_j) \in R} y_j \right)^2.$$

Мы получили, что информативность вершины измеряется её дисперсией — чем ниже разброс целевой переменной, тем лучше вершина. Разумеется, можно использовать и другие функции ошибки L — например, при выборе абсолютного отклонения мы получим в качестве критерия среднее абсолютное отклонение от медианы.

5.4.2 Классификация

Обозначим через p_k долю объектов класса k ($k \in \{1, \dots, K\}$), попавших в вершину R:

$$p_k = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} [y_i = k].$$

Через k_* обозначим класс, чьих представителей оказалось больше всего среди объектов, попавших в данную вершину: $k_* = \arg\max_k p_k$.

Ошибка классификации. Рассмотрим индикатор ошибки как функцию потерь:

$$H(R) = \min_{c \in \mathbb{Y}} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} [y_i \neq c].$$

Легко видеть, что оптимальным предсказанием тут будет наиболее популярный класс k_* — значит, критерий будет равен следующей доле ошибок:

$$H(R) = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} [y_i \neq k_*] = 1 - p_{k_*}.$$

Данный критерий является достаточно грубым, поскольку учитывает частоту p_{k_*} лишь одного класса.

Критерий Джини. Рассмотрим ситуацию, в которой мы выдаём в вершине не один класс, а распределение на всех классах $c = (c_1, \dots, c_K)$, $\sum_{k=1}^K c_k = 1$. Качество такого распределения можно измерять, например, с помощью критерия Бриера (Brier score):

$$H(R) = \min_{\sum_k c_k = 1} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} \sum_{k=1}^{K} (c_k - [y_i = k])^2.$$

Можно показать, что оптимальный вектор вероятностей состоит из долей классов p_k:

$$c_* = (p_1, \dots, p_K)$$

Если подставить эти вероятности в исходный критерий информативности и провести ряд преобразований, то мы получим критерий Джини:

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

Энтропийный критерий. Мы уже знакомы с более популярным способом оценивания качества вероятностей — логарифмическими потерями, или логарифмом правдоподобия:

$$H(R) = \min_{\sum_k c_k = 1} \left(-\frac{1}{|R|} \sum_{(x_i, y_i) \in R} \sum_{k=1}^{K} [y_i = k] \log c_k \right).$$

Для вывода оптимальных значений c_k вспомним, что все значения c_k должны суммироваться в единицу. Как известного из методов оптимизации, для учёта этого ограничения необходимо искать минимум лагранжиана:

$$L(c, \lambda) = -\frac{1}{|R|} \sum_{(x_i, y_i) \in R} \sum_{k=1}^{K} [y_i = k] \log c_k + \lambda \sum_{k=1}^{K} c_k \to \min_{c_k}$$

Дифференцируя, получаем:

$$\frac{\partial}{\partial c_k} L(c, \lambda) = -\frac{1}{|R|} \sum_{(x_i, y_i) \in R} [y_i = k] \frac{1}{c_k} + \lambda = -\frac{p_k}{c_k} + \lambda = 0,$$

откуда выражаем $c_k = p_k / \lambda$. Суммируя эти равенства по k, получим

$$1 = \sum_{k=1}^{K} c_k = \frac{1}{\lambda} \sum_{k=1}^{K} p_k = \frac{1}{\lambda},$$

откуда $\lambda = 1$. Значит, минимум достигается при $c_k = p_k$, как и в предыдущем случае. Подставляя эти выражения в критерий, получим, что он будет представлять собой энтропию распределения классов:

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

Из теории вероятностей известно, что энтропия ограничена снизу нулем, причем минимум достигается на вырожденных распределениях ($p_i = 1, p_j = 0$ для $i \neq j$). Максимальное же значение энтропия принимает для равномерного распределения. Отсюда видно, что энтропийный критерий отдает предпочтение более «вырожденным» распределениям классов в вершине.

§5.5 Критерии останова

Можно придумать большое количестве критериев останова. Перечислим некоторые ограничения и критерии:

- Ограничение максимальной глубины дерева.
- Ограничение минимального числа объектов в листе.
- Ограничение максимального количества листьев в дереве.
- Останов в случае, если все объекты в листе относятся к одному классу.
- Требование, что функционал качества при дроблении улучшался как минимум на s процентов.

С помощью грамотного выбора подобных критериев и их параметров можно существенно повлиять на качество дерева. Тем не менее, такой подбор является трудозатратным и требует проведения кросс-валидации.

§5.6 Методы стрижки дерева

Стрижка дерева является альтернативой критериям останова, описанным выше. При использовании стрижки сначала строится переобученное дерево (например, до тех пор, пока в каждом листе не окажется по одному объекту), а затем производится оптимизация его структуры с целью улучшения обобщающей способности. Существует ряд исследований, показывающих, что стрижка позволяет достичь лучшего качества по сравнению с ранним остановом построения дерева на основе различных критериев.

Тем не менее, на данный момент методы стрижки редко используются и не реализованы в большинстве библиотек для анализа данных. Причина заключается в том, что деревья сами по себе являются слабыми алгоритмами и не представляют большого интереса, а при использовании в композициях они либо *должны* быть переобучены (в случайных лесах), либо должны иметь очень небольшую глубину (в бустинге), из-за чего необходимость в стрижке отпадает.

Одним из методов стрижки является *cost-complexity pruning*. Обозначим дерево, полученное в результате работы жадного алгоритма, через T_0. Поскольку в каждом из листьев находятся объекты только одного класса, значение функционала $R(T)$ будет минимально на самом дереве T_0 (среди всех поддеревьев). Однако данный функционал характеризует лишь качество дерева на обучающей выборке, и чрезмерная подгонка под нее может привести к переобучению. Чтобы преодолеть эту проблему, введем новый функционал $R_\alpha(T)$, представляющий собой сумму исходного функционала $R(T)$ и штрафа за размер дерева:

$$R_\alpha(T) = R(T) + \alpha|T|, \tag{5.1}$$

где $|T|$ — число листьев в поддереве T, а $\alpha \geqslant 0$ — параметр. Это один из примеров *регуляризованных* критериев качества, которые ищут баланс между качеством классификации обучающей выборки и сложностью построенной модели.

Можно показать, что существует последовательность вложенных деревьев с одинаковыми корнями:

$$T_K \subset T_{K-1} \subset \cdots \subset T_0,$$

(здесь T_K — тривиальное дерево, состоящее из корня дерева T_0), в которой каждое дерево T_i минимизирует критерий (5.1) для α из интервала $\alpha \in [\alpha_i, \alpha_{i+1})$, причем

$$0 = \alpha_0 < \alpha_1 < \cdots < \alpha_K < \infty.$$

Эту последовательность можно достаточно эффективно найти путем обхода дерева. Далее из нее выбирается оптимальное дерево по отложенной выборке или с помощью кросс-валидации.

§5.7 Учет категориальных признаков

Самый очевидный способ обработки категориальных признаков — разбивать вершину на столько поддеревьев, сколько имеется возможных значений у признака (multi-way splits). Такой подход может показывать хорошие результаты, но при этом есть риск получения дерева с крайне большим числом листьев.

Рассмотрим подробнее другой подход. Пусть категориальный признак x_j имеет множество значений $Q = \{u_1, \ldots, u_q\}$, $|Q| = q$. Разобьем множество значений на два непересекающихся подмножества: $Q = Q_1 \sqcup Q_2$, и определим предикат как индикатор попадания в первое подмножество: $\beta(x) = [x_j \in Q_1]$. Таким образом, объект будет попадать в левое поддерево, если признак x_j попадает в множество Q_1, и в правое поддерево в противном случае. Основная проблема заключается в том, что для построения оптимального предиката нужно перебрать $2^{q-1} - 1$ вариантов разбиения, что может быть не вполне возможным.

Оказывается, можно обойтись без полного перебора в случаях с бинарной классификацией и регрессией [12]. Обозначим через $R_m(u)$ множество объектов, которые попали в вершину m и у которых j-й признак имеет значение u; через $N_m(u)$ обозначим количество таких объектов.

В случае с бинарной классификацией упорядочим все значения категориального признака на основе того, какая доля объектов с таким значением имеет класс $+1$:

$$\frac{1}{N_m(u_{(1)})} \sum_{x_i \in R_m(u_{(1)})} [y_i = +1] \leqslant \ldots \leqslant \frac{1}{N_m(u_{(q)})} \sum_{x_i \in R_m(u_{(q)})} [y_i = +1],$$

после чего заменим категорию $u_{(i)}$ на число i, и будем искать разбиение как для вещественного признака. Можно показать, что если искать оптимальное разбиение по критерию Джини или энтропийному критерию, то мы получим такое же разбиение, как и при переборе по всем возможным $2^{q-1} - 1$ вариантам.

Для задачи регрессии с MSE-функционалом это тоже будет верно, если упорядочивать значения признака по среднему ответу объектов с таким значением:

$$\frac{1}{N_m(u_{(1)})} \sum_{x_i \in R_m(u_{(1)})} y_i \leqslant \ldots \leqslant \frac{1}{N_m(u_{(q)})} \sum_{x_i \in R_m(u_{(q)})} y_i.$$

Именно такой подход используется в библиотеке Spark MLlib [6].

§5.8 Решающие деревья и линейные модели

Как следует из определения, решающее дерево $a(x)$ разбивает всё признаковое пространство на некоторое количество непересекающихся подмножеств $\{J_1, \ldots, J_n\}$, и в каждом подмножестве J_j выдаёт константный прогноз w_j. Значит, соответствующий алгоритм можно записать аналитически:

$$a(x) = \sum_{j=1}^{n} w_j [x \in J_j].$$

Обратим внимание, что это линейная модель над признаками $([x \in J_j])_{j=1}^n$ — а ведь в начале лекции мы хотели избавиться от линейности! Получается, что решающее дерево с помощью жадного алгоритма подбирает преобразование признаков для данной задачи, а затем просто строит линейную модель над этими признаками. Далее мы увидим, что многие нелинейные методы машинного обучения можно представить как совокупность линейных методов и хитрых способов порождения признаков.

[6] http://spark.apache.org/docs/latest/mllib-decision-tree.html

§5.9 Задачи

Задача 1. Выше говорилось, что критерий информативности для набора объектов R вычисляется на основе того, насколько хорошо их целевые переменные предсказываются константой (при оптимальном выборе этой константы):

$$H(R) = \min_{c \in \mathbb{Y}} \frac{1}{|R|} \sum_{(x_i, y_i) \in R} L(y_i, c),$$

где $L(y, c)$ — некоторая функция потерь. Соответственно, чтобы получить вид критерия при конкретной функции потерь, необходимо аналитически найти оптимальное значение константы и подставить его в формулу для $H(R)$.

Выведите критерии информативности для следующих функций потерь:

1) $L(y, c) = (y - c)^2$;

2) $L(y, c) = \sum_{k=1}^{K} (c_k - [y = k])^2$;

3) $L(y, c) = -\sum_{k=1}^{K} [y = k] \log c_k$, $\sum_{k=1}^{K} c_k = 1$.

У вас должны получиться дисперсия, критерий Джини и энтропийный критерий соответственно.

Решение.

1) Для случая $L(y, c) = (y - c)^2$ в качестве критерия информативности мы получаем MSE, а как известно, среднее — это оптимальная константа для MSE. Таким образом:

$$c = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} y_i$$

$$H(R) = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} \left(y_i - \frac{\sum_{(x_i, y_i) \in R} y_i}{|R|} \right) = \mathbb{D}[y]$$

2) Далее, считая, что $c \in \mathbb{R}^K$, найдем оптимальный вектор для критерия информативности:

$$H(R, c) = \frac{1}{|R|} \sum_{(x_i, y_i) \in R} \sum_{k=1}^{K} (c_k - [y_i = k])^2 =$$

$$= \frac{1}{|R|} \sum_{(x_i, y_i) \in R} \sum_{k=1}^{K} (c_k^2 - 2c_k [y_i = k] + [y_i = k]) =$$

$$= \frac{1}{|R|} \sum_{k=1}^{K} (|R| c_k^2 - 2c_k n_k + n_k) = \sum_{k=1}^{K} (c_k^2 - 2c_k p_k + p_k)$$

Здесь n_k - число объектов класса k, p_k - соответственно, их доля. Поскольку получилась сумма по всем классам, то прооптимизируем функционал отдельно по каждому c_k:

$$\frac{\partial H}{\partial c_k} = 2c_k - 2p_k = 0 \Rightarrow c_k = p_k$$

Очевидно, при $c = (p_1, ..., p_k)$ достигается минимум функционала. Подставим это значение и посчитаем критерий информативности:

$$H(R) = \sum_{k=1}^{K}(p_k^2 - 2p_k^2 + p_k) = \sum_{k=1}^{K}(p_k - p_k^2) = 1 - \sum_{k=1}^{K} p_k^2$$

Это и есть не что иное, как критерий Джини.

3) Для начала подставим функцию потерь в критерий информативности и запишем оптимизационную задачу. Первый шаг аналогичен первому пункту:

$$H(R, c) = -\frac{1}{|R|} \sum_{(x_i, y_i) \in R} \sum_{k=1}^{K} [y_i = k] \log c_k = -\sum_{k=1}^{K} p_k \log c_k$$

Задача оптимизации:

$$\begin{cases} -\sum_{k=1}^{K} p_k \log c_k \to \min_{c_k} \\ \sum_{k=1}^{K} c_k = 1 \\ c_k > 0, k = 1...K \end{cases}$$

Решим ее с помощью метода Лагранжа для условного экстремума:

$$L(c) = -\sum_{k=1}^{K} p_k \log c_k + \lambda \left(\sum_{k=1}^{K} c_k - 1 \right)$$

$$\begin{cases} \frac{\partial L}{\partial c_k} = -\frac{p_k}{c_k} + \lambda = 0 \\ \varphi(c) = \sum_{k=1}^{K} c_k - 1 = 0 \end{cases}$$

$$\begin{cases} c_k = \frac{p_k}{\lambda} \\ \sum_{k=1}^{K} \frac{p_k}{\lambda} = 1 \end{cases}$$

$$\begin{cases} \lambda = \sum_{k=1}^{K} p_k = 1 \\ c_k = p_k \end{cases}$$

Таким образом:

$$H(R) = -\sum_{k=1}^{K} p_k \log p_k$$

А это и есть энтропийный критерий.

6 Ансамблевые методы обучения

При обсуждении решающих деревьев мы упомянули, что они могут восстанавливать очень сложные закономерности, но при этом неустойчивы к малейшим изменениям в данных. Из-за этого сами по себе деревья не очень хороши, но при этом, как оказывается, при объединении в *композицию* они показывают очень хорошие результаты. Одним из подходов к построению композиций является *бэггинг*, который независимо строит несколько моделей и усредняет их ответы. Сначала изучим инструмент, который поможет нам в анализе бэггинга — декомпозицию ошибки на компоненты смещения и разброса (bias-variance decomposition) — а затем перейдем к самим методам. Также существует другой подход к построению композиций, называемый *бустингом*, который строит модели последовательно, и каждая следующая модель исправляет ошибки предыдущей.

§6.1 Вопросы для самопроверки

1. Что такое бутстрап?
2. Для какой ошибки строится разложение на шум, смещение и разброс? Запишите формулу этой ошибки.
3. Приведите пример семейства алгоритмов с низким смещением и большим разбросом; семейства алгоритмов с большим смещением и низким разбросом. Поясните примеры.
4. Что такое бэггинг? Какие у него преимущества? Какие недостатки?
5. Что такое случайный лес? Чем он отличается от бэггинга над решающими деревьями?
6. За счет чего в случайных лесах понижается корреляция между базовыми алгоритмами?
7. Что такое out-of-bag оценка в бэггинге?
8. Опишите идею градиентного бустинга для среднеквадратичной ошибки. Запишите задачу для обучения очередной базовой модели.
9. Опишите идею градиентного бустинга для произвольной функции потерь. Запишите задачу для обучения очередной базовой модели.
10. Запишите вид композиции, которая обучается в градиентном бустинге. Как выбирают количество базовых алгоритмов в ней?
11. Что такое сдвиги в градиентном бустинге? Как они вычисляются и для чего используются?
12. Как обучается очередной базовый алгоритм в градиентном бустинге? Что такое сокращение шага?

13. Какие решающие деревья используются в случайных лесах? Сколько нужно базовых моделей для случайного леса? (6.6.5)

14. Какие решающие деревья используются в градиентном бустинге? Сколько нужно базовых моделей для градиентного бустинга? (6.6.5)

15. В чём заключается переподбор прогнозов в листьях решающих деревьев в градиентном бустинге?

16. Как в xgboost выводится функционал ошибки с помощью разложения в ряд Тейлора?

17. Какие регуляризации используются в xgboost?

§6.2 Бутстрап

Рассмотрим простой пример построения композиции алгоритмов. Пусть дана конечная выборка $X = (x_i, y_i)$ с вещественными ответами. Будем решать задачу линейной регрессии. Сгенерируем подвыборку с помощью *бутстрапа*. Равномерно возьмем из выборки ℓ объектов с возвращением. Отметим, что из-за возвращения среди них окажутся повторы. Обозначим новую выборку через X_1. Повторив процедуру N раз, сгенерируем N подвыборок $X_1, \ldots, X_N$. Обучим по каждой из них линейную модель регрессии, получив *базовые алгоритмы* $b_1(x), \ldots, b_N(x)$.

Предположим, что существует истинная функция ответа для всех объектов $y(x)$, а также задано распределение на объектах $p(x)$. В этом случае мы можем записать ошибку каждой функции регрессии

$$\varepsilon_j(x) = b_j(x) - y(x), \qquad j = 1, \ldots, N,$$

и записать матожидание среднеквадратичной ошибки

$$\mathbb{E}_x(b_j(x) - y(x))^2 = \mathbb{E}_x\varepsilon_j^2(x).$$

Средняя ошибка построенных функций регрессии имеет вид

$$E_1 = \frac{1}{N}\sum_{j=1}^{N}\mathbb{E}_x\varepsilon_j^2(x).$$

Предположим, что ошибки несмещены и некоррелированы:

$$\mathbb{E}_x\varepsilon_j(x) = 0;$$
$$\mathbb{E}_x\varepsilon_i(x)\varepsilon_j(x) = 0, \quad i \neq j.$$

Построим теперь новую функцию регрессии, которая будет усреднять ответы построенных нами функций:

$$a(x) = \frac{1}{N}\sum_{j=1}^{N} b_j(x).$$

Найдем ее среднеквадратичную ошибку:

$$\begin{aligned} E_N &= \mathbb{E}_x \left(\frac{1}{N} \sum_{j=1}^{n} b_j(x) - y(x) \right)^2 = \\ &= \mathbb{E}_x \left(\frac{1}{N} \sum_{j=1}^{N} \varepsilon_j(x) \right)^2 = \\ &= \frac{1}{N^2} \mathbb{E}_x \left(\sum_{j=1}^{N} \varepsilon_j^2(x) + \underbrace{\sum_{i \neq j} \varepsilon_i(x)\varepsilon_j(x)}_{=0} \right) = \\ &= \frac{1}{N} E_1. \end{aligned}$$

Таким образом, усреднение ответов позволило уменьшить средний квадрат ошибки в N раз!

Следует отметить, что рассмотренный нами пример не очень применим на практике, поскольку мы сделали предположение о некоррелированности ошибок, что редко выполняется. Если это предположение неверно, то уменьшение ошибки оказывается не таким значительным. Позже мы рассмотрим более сложные методы объединения алгоритмов в композицию, которые позволяют добиться высокого качества в реальных задачах.

§6.3 Bias-Variance decomposition

Оказывается, ошибка любой модели складывается из трех факторов: сложности самой выборки, сходства модели с истинной зависимостью ответов от объектов в выборке, и богатства семейства, из которого выбирается конкретная модель. Между этими факторами существует некоторый баланс, и уменьшение одного из них приводит к увеличению другого. Такое разложение ошибки носит название разложения на смещение и разброс, и его формальным выводом мы сейчас займемся.

6.3.1 Минимум среднеквадратичного риска

Пусть задана выборка $X = (x_i, y_i)_{i=1}^{\ell}$ с вещественными ответами $y_i \in \mathbb{R}$ (рассматриваем задачу регрессии). Будем считать, что на пространстве всех объектов и ответов $\mathbb{X} \times \mathbb{Y}$ существует распределение $p(x, y)$, из которого сгенерирована выборка X и ответы на ней.

Рассмотрим квадратичную функцию потерь

$$L(y, a) = \big(y - a(x)\big)^2$$

и соответствующий ей *среднеквадратичный риск*

$$R(a) = \mathbb{E}_{x,y}\Big[\big(y - a(x)\big)^2\Big] = \int_{\mathbb{X}} \int_{\mathbb{Y}} p(x, y)\big(y - a(x)\big)^2 dxdy.$$

Данный функционал усредняет ошибку модели в каждой точке пространства x и для каждого возможного ответа y, причём вклад пары (x, y), по сути, пропорционален

вероятности получить её в выборке $p(x, y)$. Разумеется, на практике мы не можем вычислить данный функционал, поскольку распределение $p(x, y)$ неизвестно. Тем не менее, в теории он позволяет измерить качество модели на всех возможных объектах, а не только на обучающей выборке.

Покажем, что минимум среднеквадратичного риска достигается на функции, возвращающей условное матожидание ответа при фиксированном объекте:

$$a_*(x) = \mathbb{E}[y \mid x] = \int_{\mathbb{Y}} yp(y \mid x)dy = \arg\min_a R(a).$$

Преобразуем функцию потерь:

$$\begin{aligned} L(y, a(x)) &= (y - a(x))^2 = (y - \mathbb{E}(y \mid x) + \mathbb{E}(y \mid x) - a(x))^2 = \\ &= (y - \mathbb{E}(y \mid x))^2 + 2\big(y - \mathbb{E}(y \mid x)\big)\big(\mathbb{E}(y \mid x) - a(x)\big) + (\mathbb{E}(y \mid x) - a(x))^2. \end{aligned}$$

Подставляя ее в функционал среднеквадратичного риска, получаем:

$$\begin{aligned} R(a) &= \mathbb{E}_{x,y} L(y, a(x)) = \\ &= \mathbb{E}_{x,y}(y - \mathbb{E}(y \mid x))^2 + \mathbb{E}_{x,y}(\mathbb{E}(y \mid x) - a(x))^2 + \\ &+ 2\mathbb{E}_{x,y}\big(y - \mathbb{E}(y \mid x)\big)\big(\mathbb{E}(y \mid x) - a(x)\big). \end{aligned}$$

Разберемся сначала с последним слагаемым. Перейдём от матожидания $\mathbb{E}_{x,y}[f(x, y)]$ к цепочке матожиданий

$$\mathbb{E}_x\mathbb{E}_y[f(x, y) \mid x] = \int_{\mathbb{X}} \left(\int_{\mathbb{Y}} f(x, y)p(y \mid x)dy \right) p(x)dx$$

и заметим, что величина $\big(\mathbb{E}(y \mid x) - a(x)\big)$ не зависит от y, и поэтому ее можно вынести за матожидание по y:

$$\begin{aligned} &\mathbb{E}_x\mathbb{E}_y\Big[\big(y - \mathbb{E}(y \mid x)\big)\big(\mathbb{E}(y \mid x) - a(x)\big) \mid x\Big] = \\ &= \mathbb{E}_x\Big(\big(\mathbb{E}(y \mid x) - a(x)\big)\mathbb{E}_y\Big[\big(y - \mathbb{E}(y \mid x)\big) \mid x\Big]\Big) = \\ &= \mathbb{E}_x\Big(\big(\mathbb{E}(y \mid x) - a(x)\big)\big(\mathbb{E}(y \mid x) - \mathbb{E}(y \mid x)\big)\Big) = \\ &= 0 \end{aligned}$$

Получаем, что функционал среднеквадратичного риска имеет вид

$$R(a) = \mathbb{E}_{x,y}(y - \mathbb{E}(y \mid x))^2 + \mathbb{E}_{x,y}(\mathbb{E}(y \mid x) - a(x))^2.$$

От алгоритма $a(x)$ зависит только второе слагаемое, и оно достигает своего минимума, если $a(x) = \mathbb{E}(y \mid x)$. Таким образом, оптимальная модель регрессии для квадратичной функции потерь имеет вид

$$a_*(x) = \mathbb{E}(y \mid x) = \int_{\mathbb{Y}} yp(y \mid x)dy.$$

Иными словами, мы должны провести «взвешенное голосование» по всем возможным ответам, причем вес ответа равен его апостериорной вероятности.

6.3.2 Ошибка метода обучения

Для того, чтобы построить идеальную функцию регрессии, необходимо знать распределение на объектах и ответах $p(x, y)$, что, как правило, невозможно. На практике вместо этого выбирается некоторый *метод обучения* $\mu : (\mathbb{X} \times \mathbb{Y})^\ell \to \mathcal{A}$, который произвольной обучающей выборке ставит в соответствие некоторый алгоритм из семейства $\mathcal{A}$. В качестве меры качества метода обучения можно взять усредненный по всем выборкам среднеквадратичный риск алгоритма, выбранного методом μ по выборке:

$$L(\mu) = \mathbb{E}_X\Big[\mathbb{E}_{x,y}\Big[\big(y - \mu(X)(x)\big)^2\Big]\Big] = \tag{6.1}$$
$$= \int_{(\mathbb{X}\times\mathbb{Y})^\ell} \int_{\mathbb{X}\times\mathbb{Y}} \big(y - \mu(X)(x)\big)^2 p(x,y) \prod_{i=1}^{\ell} p(x_i, y_i) dx dy dx_1 dy_1 \ldots dx_\ell dy_\ell.$$

Здесь матожидание $\mathbb{E}_X[\cdot]$ берется по всем возможным выборкам $\{(x_1, y_1), \ldots, (x_\ell, y_\ell)\}$ из распределения $\prod_{i=1}^{\ell} p(x_i, y_i)$.

Обратим внимание, что результатом применения метода обучения $\mu(X)$ к выборке X является модель, поэтому правильно писать $\mu(X)(x)$. Но это довольно громоздкая запись, поэтому будем везде дальше писать просто $\mu(X)$, но не будем забывать, что это функция, зависящая от объекта x.

Выше мы показали, что среднеквадратичный риск на фиксированной выборке X можно расписать как

$$\mathbb{E}_{x,y}\Big[\big(y - \mu(X)\big)^2\Big] = \mathbb{E}_{x,y}\Big[\big(y - \mathbb{E}[y \mid x]\big)^2\Big] + \mathbb{E}_{x,y}\Big[\big(\mathbb{E}[y \mid x] - \mu(X)\big)^2\Big].$$

Подставим это представление в (6.1):

$$L(\mu) = \mathbb{E}_X\Big[\underbrace{\mathbb{E}_{x,y}\Big[\big(y - \mathbb{E}[y \mid x]\big)^2\Big]}_{\text{не зависит от } X} + \mathbb{E}_{x,y}\Big[\big(\mathbb{E}[y \mid x] - \mu(X)\big)^2\Big]\Big] =$$
$$= \mathbb{E}_{x,y}\Big[\big(y - \mathbb{E}[y \mid x]\big)^2\Big] + \mathbb{E}_{x,y}\Big[\mathbb{E}_X\Big[\big(\mathbb{E}[y \mid x] - \mu(X)\big)^2\Big]\Big]. \tag{6.2}$$

Преобразуем второе слагаемое:

$$\mathbb{E}_{x,y}\Big[\mathbb{E}_X\Big[\big(\mathbb{E}[y \mid x] - \mu(X)\big)^2\Big]\Big] =$$
$$= \mathbb{E}_{x,y}\Big[\mathbb{E}_X\Big[\big(\mathbb{E}[y \mid x] - \mathbb{E}_X[\mu(X)] + \mathbb{E}_X[\mu(X)] - \mu(X)\big)^2\Big]\Big] =$$
$$= \mathbb{E}_{x,y}\Big[\mathbb{E}_X\Big[\underbrace{\big(\mathbb{E}[y \mid x] - \mathbb{E}_X[\mu(X)]\big)^2}_{\text{не зависит от } X}\Big]\Big] + \mathbb{E}_{x,y}\Big[\mathbb{E}_X\Big[\big(\mathbb{E}_X[\mu(X)] - \mu(X)\big)^2\Big]\Big] +$$
$$+ 2\mathbb{E}_{x,y}\Big[\mathbb{E}_X\Big[\big(\mathbb{E}[y \mid x] - \mathbb{E}_X[\mu(X)]\big)\big(\mathbb{E}_X[\mu(X)] - \mu(X)\big)\Big]\Big]. \tag{6.3}$$

Покажем, что последнее слагаемое обращается в нуль:

$$\mathbb{E}_X\Big[\big(\mathbb{E}[y \mid x] - \mathbb{E}_X[\mu(X)]\big)\big(\mathbb{E}_X[\mu(X)] - \mu(X)\big)\Big] =$$
$$= \big(\mathbb{E}[y \mid x] - \mathbb{E}_X[\mu(X)]\big)\mathbb{E}_X\Big[\mathbb{E}_X[\mu(X)] - \mu(X)\Big] =$$
$$= \big(\mathbb{E}[y \mid x] - \mathbb{E}_X[\mu(X)]\big)\Big[\mathbb{E}_X[\mu(X)] - \mathbb{E}_X[\mu(X)]\Big] =$$
$$= 0.$$

Учитывая это, подставим (6.3) в (6.2):

$$L(\mu) = \underbrace{\mathbb{E}_{x,y}\Big[\big(y - \mathbb{E}[y \,|\, x]\big)^2\Big]}_{\text{шум}} + \\ + \underbrace{\mathbb{E}_x\Big[\big(\mathbb{E}_X\big[\mu(X)\big] - \mathbb{E}[y \,|\, x]\big)^2\Big]}_{\text{смещение}} + \underbrace{\mathbb{E}_x\Big[\mathbb{E}_X\Big[\big(\mu(X) - \mathbb{E}_X\big[\mu(X)\big]\big)^2\Big]\Big]}_{\text{разброс}}. \tag{6.4}$$

Рассмотрим подробнее компоненты полученного разложения ошибки. Первая компонента характеризует *шум* в данных и равна ошибке идеального алгоритма. Невозможно построить алгоритм, имеющий меньшую среднеквадратичную ошибку. Вторая компонента характеризует *смещение (bias)* метода обучения, то есть отклонение среднего ответа обученного алгоритма от ответа идеального алгоритма. Третья компонента характеризует *дисперсию (variance)*, то есть разброс ответов обученных алгоритмов относительно среднего ответа.

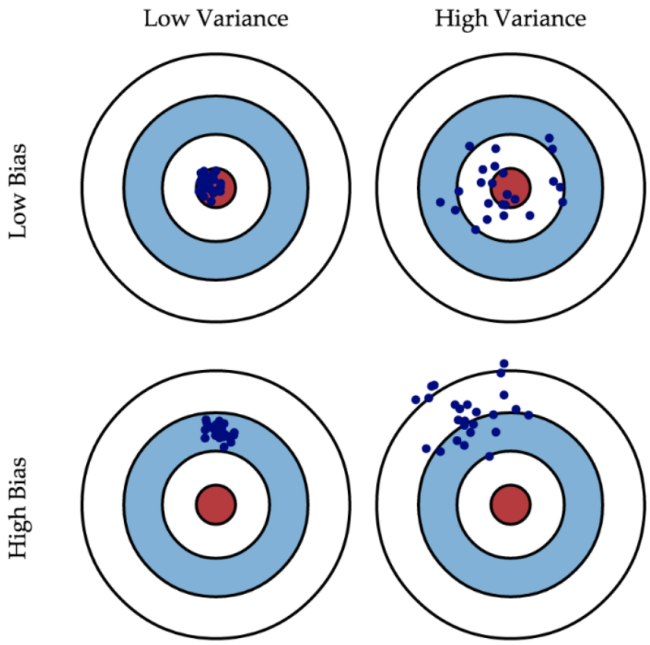

Рис. 8. Иллюстрация сдвига и разброса для различных моделей.

Разложение для произвольной функции потерь. Разложение ошибки на три компоненты, которое мы только что вывели, верно только для квадратичной функции потерь. Существуют более общие формы этого разложения [14], которые состоят из трёх компонент с аналогичным смыслом, поэтому можно утверждать, что для большинства распространённых функций потерь ошибка метода обучения складывается из шума, смещения и разброса; значит, и дальнейшие рассуждения про изменение этих компонент в композициях также можно обобщить на другие функции потерь (например, на индикатор ошибки классификации).

Смещение показывает, насколько хорошо с помощью данных метода обучения и семейства алгоритмов можно приблизить оптимальный алгоритм. Как правило,

смещение маленькое у сложных семейств (например, у деревьев) и большое у простых семейств (например, линейных классификаторов). Дисперсия показывает, насколько сильно может изменяться ответ обученного алгоритма в зависимости от выборки — иными словами, она характеризует чувствительность метода обучения к изменениям в выборке. Как правило, простые семейства имеют маленькую дисперсию, а сложные семейства — большую дисперсию.

На рис. 8 изображены модели с различными сдвигом и разбросом. Модели изображены синими точками, одна точка соответствует модели, обученной по одной из возможных обучающих выборок. Каждый круг характеризует качество модели — чем ближе точка к центру, тем меньше ошибок на контрольной выборке достигает данный алгоритм. Видно, что большой сдвиг соответствует тому, что в среднем точки не попадают в центр, то есть в среднем они не соответствуют лучшей модели. Большой разброс означает, что модель может попасть по качеству куда угодно — как в центр, так и в область с большой ошибкой.

§6.4 Бэггинг

Пусть имеется некоторый метод обучения $\mu(X)$. Построим на его основе метод $\tilde{\mu}(X)$, который генерирует случайную подвыборку $\tilde{X}$ с помощью бутстрапа и подает ее на вход метода μ: $\tilde{\mu}(X) = \mu(\tilde{X})$. Напомним, что бутстрап представляет собой сэмплирование ℓ объектов из выборки с возвращением, в результате чего некоторые объекты выбираются несколько раз, а некоторые — ни разу. Помещение нескольких копий одного объекта в бутстрапированную выборку соответствует выставлению веса при данном объекте — соответствующее ему слагаемое несколько раз войдет в функционал, и поэтому штраф за ошибку на нем будет больше.

В *бэггинге (bagging, bootstrap aggregation)* предлагается обучить некоторое число алгоритмов $b_n(x)$ с помощью метода $\tilde{\mu}$, и построить итоговую композицию как среднее данных базовых алгоритмов:

$$a_N(x) = \frac{1}{N}\sum_{n=1}^{N} b_n(x) = \frac{1}{N}\sum_{n=1}^{N} \tilde{\mu}(X)(x).$$

Заметим, что в методе обучения для бэггинга появляется ещё один источник случайности — взятие подвыборки. Чтобы функционал качества $L(\mu)$ был детерминированным, мы будем далее считать, что матожидание $\mathbb{E}_X[\cdot]$ берётся не только по всем обучающим выборкам X, но ещё и по всем возможным подвыборкам $\tilde{X}$, получаемым с помощью бутстрапа. Это вполне логичное обобщение, поскольку данное матожидание вводится в функционал именно для учёта случайностей, связанных с процедурой обучения модели.

Найдём смещение из разложения (6.4) для бэггинга:

$$\begin{aligned}
&\mathbb{E}_{x,y}\Big[\Big(\mathbb{E}_X\Big[\frac{1}{N}\sum_{n=1}^{N}\tilde{\mu}(X)(x)\Big] - \mathbb{E}[y\,|\,x]\Big)^2\Big] = \\
&= \mathbb{E}_{x,y}\Big[\Big(\frac{1}{N}\sum_{n=1}^{N}\mathbb{E}_X[\tilde{\mu}(X)(x)] - \mathbb{E}[y\,|\,x]\Big)^2\Big] = \\
&= \mathbb{E}_{x,y}\Big[\big(\mathbb{E}_X\big[\tilde{\mu}(X)(x)\big] - \mathbb{E}[y\,|\,x]\big)^2\Big].
\end{aligned}$$

Мы получили, что смещение композиции, полученной с помощью бэггинга, совпадает со смещением одного базового алгоритма. Таким образом, бэггинг не ухудшает смещенность модели.

Теперь перейдём к разбросу. Запишем выражение для дисперсии композиции, обученной с помощью бэггинга:

$$\mathbb{E}_{x,y}\Big[\mathbb{E}_X\Big[\Big(\frac{1}{N}\sum_{n=1}^{N}\tilde{\mu}(X)(x) - \mathbb{E}_X\Big[\frac{1}{N}\sum_{n=1}^{N}\tilde{\mu}(X)(x)\Big]\Big)^2\Big]\Big].$$

Рассмотрим выражение, стоящее под матожиданиями:

$$\begin{aligned}
&\Big(\frac{1}{N}\sum_{n=1}^{N}\tilde{\mu}(X)(x) - \mathbb{E}_X\Big[\frac{1}{N}\sum_{n=1}^{N}\tilde{\mu}(X)(x)\Big]\Big)^2 = \\
&\quad= \frac{1}{N^2}\Big(\sum_{n=1}^{N}\Big[\tilde{\mu}(X)(x) - \mathbb{E}_X\big[\tilde{\mu}(X)(x)\big]\Big]\Big)^2 = \\
&\quad= \frac{1}{N^2}\sum_{n=1}^{N}\Big(\tilde{\mu}(X)(x) - \mathbb{E}_X\big[\tilde{\mu}(X)(x)\big]\Big)^2 + \\
&\qquad+ \frac{1}{N^2}\sum_{n_1\neq n_2}\Big(\tilde{\mu}(X)(x) - \mathbb{E}_X\big[\tilde{\mu}(X)(x)\big]\Big)\Big(\tilde{\mu}(X)(x) - \mathbb{E}_X\big[\tilde{\mu}(X)(x)\big]\Big)
\end{aligned}$$

Возьмем теперь матожидания от этого выражения, учитывая, что все базовые алгоритмы одинаково распределены относительно X:

$$\begin{aligned}
&\mathbb{E}_{x,y}\Big[\mathbb{E}_X\Big[\frac{1}{N^2}\sum_{n=1}^{N}\Big(\tilde{\mu}(X)(x) - \mathbb{E}_X\big[\tilde{\mu}(X)(x)\big]\Big)^2 + \\
&\qquad+ \frac{1}{N^2}\sum_{n_1\neq n_2}\Big(\tilde{\mu}(X)(x) - \mathbb{E}_X\big[\tilde{\mu}(X)(x)\big]\Big)\Big(\tilde{\mu}(X)(x) - \mathbb{E}_X\big[\tilde{\mu}(X)(x)\big]\Big)\Big]\Big] = \\
&= \frac{1}{N^2}\mathbb{E}_{x,y}\Big[\mathbb{E}_X\Big[\sum_{n=1}^{N}\Big(\tilde{\mu}(X)(x) - \mathbb{E}_X\big[\tilde{\mu}(X)(x)\big]\Big)^2\Big]\Big] + \\
&\qquad+ \frac{1}{N^2}\mathbb{E}_{x,y}\Big[\mathbb{E}_X\Big[\sum_{n_1\neq n_2}\Big(\tilde{\mu}(X)(x) - \mathbb{E}_X\big[\tilde{\mu}(X)(x)\big]\Big)\times \\
&\qquad\qquad\times\Big(\tilde{\mu}(X)(x) - \mathbb{E}_X\big[\tilde{\mu}(X)(x)\big]\Big)\Big]\Big] = \\
&= \frac{1}{N}\mathbb{E}_{x,y}\Big[\mathbb{E}_X\Big[\Big(\tilde{\mu}(X)(x) - \mathbb{E}_X\big[\tilde{\mu}(X)(x)\big]\Big)^2\Big]\Big] + \\
&\qquad+ \frac{N(N-1)}{N^2}\mathbb{E}_{x,y}\Big[\mathbb{E}_X\Big[\Big(\tilde{\mu}(X)(x) - \mathbb{E}_X\big[\tilde{\mu}(X)(x)\big]\Big)\times \\
&\qquad\qquad\times\Big(\tilde{\mu}(X)(x) - \mathbb{E}_X\big[\tilde{\mu}(X)(x)\big]\Big)\Big]\Big]
\end{aligned}$$

Первое слагаемое — это дисперсия одного базового алгоритма, деленная на длину композиции N. Второе — ковариация между двумя базовыми алгоритмами. Мы видим, что *если базовые алгоритмы некоррелированы, то дисперсия композиции в N раз меньше дисперсии отдельных алгоритмов*. Если же корреляция имеет место, то уменьшение дисперсии может быть гораздо менее существенным.

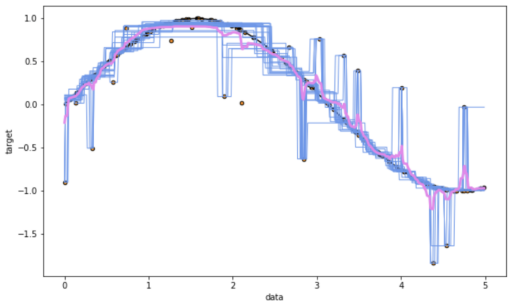

Рис. 9. Использование композиции решающих деревьев на обычной выборке в задаче регрессии.

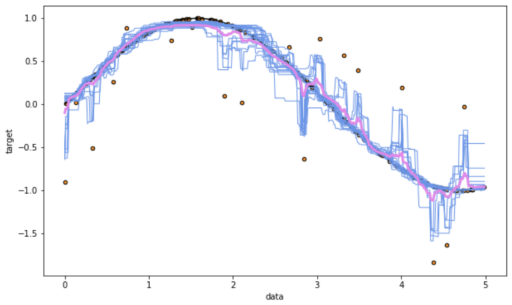

Рис. 10. Использование композиции решающих деревьев на бустрапированной подвыборке в задаче регрессии.

§6.5 Случайный лес

Как мы выяснили, бэггинг позволяет объединить несмещенные, но чувствительные к обучающей выборке алгоритмы в несмещенную композицию с низкой дисперсией. Хорошим семейством базовых алгоритмов здесь являются решающие деревья — они достаточно сложны и могут достигать нулевой ошибки на любой выборке (следовательно, имеют низкое смещение), но в то же время легко переобучаются.

Алгоритм 6.1. Random Forest

1: **для** $n = 1, \ldots, N$
2: Сгенерировать выборку $\tilde{X}_n$ с помощью бутстрэпа
3: Построить решающее дерево $b_n(x)$ по выборке $\tilde{X}_n$:
- дерево строится, пока в каждом листе не окажется не более $n_{\min}$ объектов
- при каждом разбиении сначала выбирается m случайных признаков из p, и оптимальное разделение ищется только среди них

4: Вернуть композицию $a_N(x) = \frac{1}{N}\sum_{n=1}^{N} b_n(x)$

Метод *случайных лесов* [15] основан на бэггинге над решающими деревьями, см. алгоритм 6.1. Выше мы отметили, что бэггинг сильнее уменьшает дисперсию базовых алгоритмов, если они слабо коррелированы. В случайных лесах корреляция между деревьями понижается путем рандомизации по двум направлениям: по объектам и по признакам. Во-первых, каждое дерево обучается по бутстрапированной подвыборке. Во-вторых, в каждой вершине разбиение ищется по подмножеству признаков. Вспомним, что при построении дерева последовательно происходит разделение вершин до тех пор, пока не будет достигнуто идеальное качество на обучении. Каждая вершина разбивает выборку по одному из признаков относительно некоторого порога. В случайных лесах признак, по которому производится разбиение, выбирается не из всех возможных признаков, а лишь из их случайного подмножества размера m.

Рекомендуется в задачах классификации брать $m = \lfloor\sqrt{d}\rfloor$, а в задачах регрессии — $m = \lfloor d/3\rfloor$, где d — число признаков. Также рекомендуется в задачах классификации строить каждое дерево до тех пор, пока в каждом листе не окажется по

одному объекту, а в задачах регрессии — пока в каждом листе не окажется по пять объектов.

Случайные леса — один из самых сильных методов построения композиций. На практике он может работать немного хуже градиентного бустинга, но при этом он гораздо более прост в реализации.

6.5.1 Out-of-Bag

Каждое дерево в случайном лесе обучается по подмножеству объектов. Это значит, что те объекты, которые не вошли в бутстрапированную выборку X_n дерева b_n, по сути являются контрольными для данного дерева. Значит, мы можем для каждого объекта x_i найти деревья, которые были обучены без него, и вычислить по их ответам out-of-bag-ошибку:

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

где $L(y, z)$ — функция потерь. Можно показать, что по мере увеличения числа деревьев N данная оценка стремится к leave-one-out-оценке, но при этом существенно проще для вычисления.

6.5.2 Связь с метрическими методами

Случайные леса, по сути, осуществляют предсказание для объекта на основе меток похожих объектов из обучения. Схожесть объектов при этом тем выше, чем чаще эти объекты оказываются в одном и том же листе дерева. Покажем это формально.

Рассмотрим задачу регрессии с квадратичной функцией потерь. Пусть $T_n(x)$ — номер листа n-го дерева из случайного леса, в который попадает объект x. Ответ дерева на объекте x равен среднему ответу по всем обучающим объектам, которые попали в лист $T_n(x)$. Это можно записать как

$$b_n(x) = \sum_{i=1}^{\ell} w_n(x, x_i) y_i,$$

где

$$w_n(x, x_i) = \frac{[T_n(x) = T_n(x_i)]}{\sum_{j=1}^{\ell}[T_n(x) = T_n(x_j)]}.$$

Тогда ответ композиции равен

$$a_N(x) = \frac{1}{N} \sum_{n=1}^{N} \sum_{i=1}^{\ell} w_n(x, x_i) y_i = \sum_{i=1}^{\ell} \left(\frac{1}{N} \sum_{n=1}^{N} w_n(x, x_i)\right) y_i.$$

Видно, что ответ случайного леса представляет собой сумму ответов всех объектов обучения с некоторыми весами, причём данные веса измеряют сходство объектов x и x_i на основе того, сколько раз они оказались в одном и том же листе. Таким образом, случайный лес позволяет ввести некоторую функцию расстояния на объектах.

Как мы узнаем позже, на этом принципе основан целый класс *метрических* методов, наиболее популярным представителем которых является метод k ближайших соседей.

Отметим, что номер листа $T_n(x)$, в который попал объект, сам по себе является ценным признаком. Достаточно неплохо работает подход, в котором по выборке обучается композиция из небольшого числа деревьев с помощью случайного леса или градиентного бустинга, а затем к ней добавляются категориальные признаки $T_1(x), T_2(x), \ldots, T_N(x)$. Новые признаки являются результатом нелинейного разбиения пространства и несут в себе информацию о сходстве объектов.

§6.6 Градиентный бустинг

Ранее мы изучили бэггинг и случайные леса — подходы к построению композиций, которые независимо обучают каждый базовый алгоритм по некоторому подмножеству обучающих данных. Бэггинг имеет пару проблем: если базовая модель окажется смещенной, то и композиция моделей не справится с поставленной задачей; базовые модели долго обучать и применять, долго хранить Возникает ощущение, что мы используем возможности объединения алгоритмов не в полную силу, и можно было бы строить их так, чтобы *каждая следующая модель исправляла ошибки предыдущих*. Ниже мы рассмотрим метод, который реализует эту идею — градиентный бустинг, предложенный Фридманом [16]. Он работает для любых дифференцируемых функций потерь и является одним из наиболее мощных и универсальных на сегодняшний день.

6.6.1 Бустинг в задаче регрессии

Рассмотрим задачу минимизации квадратичного функционала:

$$\frac{1}{2}\sum_{i=1}^{\ell}(a(x_i) - y_i)^2 \to \min_a$$

Будем искать итоговый алгоритм в виде суммы *базовых моделей* (weak learners) $b_n(x)$:

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

где базовые алгоритмы b_n принадлежат некоторому семейству $\mathcal{A}$.

Построим первый базовый алгоритм:

$$b_1(x) := \arg\min_{b\in\mathcal{A}} \frac{1}{2}\sum_{i=1}^{\ell}(b(x_i) - y_i)^2$$

Решение такой задачи не представляет трудностей для многих семейств алгоритмов. Теперь мы можем посчитать остатки на каждом объекте — расстояния от ответа нашего алгоритма до истинного ответа:

$$s_i^{(1)} = y_i - b_1(x_i)$$

Если прибавить эти остатки к ответам построенного алгоритма, то он не будет допускать ошибок на обучающей выборке. Значит, будет разумно построить второй алгоритм так, чтобы его ответы были как можно ближе к остаткам:

$$b_2(x) := \arg\min_{b \in \mathcal{A}} \frac{1}{2} \sum_{i=1}^{\ell} (b(x_i) - s_i^{(1)})^2$$

Каждый следующий алгоритм тоже будем настраивать на остатки предыдущих:

$$s_i^{(N)} = y_i - \sum_{n=1}^{N-1} b_n(x_i) = y_i - a_{N-1}(x_i), \qquad i = 1, \ldots, \ell;$$

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

Описанный метод прост в реализации, хорошо работает и может быть найден во многих библиотеках — например, в `scikit-learn`.

Заметим, что остатки могут быть найдены как антиградиент функции потерь по ответу модели, посчитанный в точке ответа уже построенной композиции:

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

Получается, что выбирается такой базовый алгоритм, который как можно сильнее уменьшит ошибку композиции — это свойство вытекает из его близости к антиградиенту функционала на обучающей выборке. Попробуем разобраться с этим свойством подробнее, а также попытаемся обобщить его на другие функции потерь.

6.6.2 Градиентный бустинг как градиентный спуск в функциональном пространстве

Пусть дана некоторая дифференцируемая функция потерь $L(y, z)$. Будем строить взвешенную сумму базовых алгоритмов:

$$a_N(x) = \sum_{n=0}^{N} \gamma_n b_n(x)$$

Заметим, что в композиции имеется начальный алгоритм $b_0(x)$. Как правило, коэффициент γ_0 при нем берут равным единице, а сам алгоритм выбирают очень простым, например:

- нулевым $b_0(x) = 0$;
- возвращающим самый популярный класс (в задачах классификации):

$$b_0(x) = \arg\max_{y \in \mathbb{Y}} \sum_{i=1}^{\ell} [y_i = y]$$

- возвращающим средний ответ (в задачах регрессии):

$$b_0(x) = \frac{1}{\ell}\sum_{i=1}^{\ell} y_i$$

Допустим, мы построили композицию $a_{N-1}(x)$ из $N-1$ алгоритма, и хотим выбрать следующий базовый алгоритм $b_N(x)$ так, чтобы как можно сильнее уменьшить ошибку:

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma_N b_N(x_i)) \to \min_{b_N, \gamma_N}$$

Ответим в первую очередь на следующий вопрос: если бы в качестве алгоритма $b_N(x)$ мы могли выбрать совершенно любую функцию, то какие значения ей следовало бы принимать на объектах обучающей выборки? Иными словами, нам нужно понять, какие числа $s_1, \ldots, s_\ell$ надо выбрать для решения следующей задачи:

$$\sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + s_i) \to \min_{s_1,\ldots,s_\ell}$$

Понятно, что можно требовать $s_i = y_i - a_{N-1}(x_i)$, но такой подход никак не учитывает особенностей функции потерь $L(y, z)$ и требует лишь точного совпадения предсказаний и истинных ответов. Более разумно потребовать, чтобы сдвиг s_i был противоположен производной функции потерь в точке $z = a_{N-1}(x_i)$:

$$s_i = -\left.\frac{\partial L}{\partial z}\right|_{z=a_{N-1}(x_i)}$$

В этом случае мы сдвинемся в сторону скорейшего убывания функции потерь. Заметим, что вектор сдвигов $s = (s_1, \ldots, s_\ell)$ совпадает с антиградиентом:

$$\left(-\left.\frac{\partial L}{\partial z}\right|_{z=a_{N-1}(x_i)}\right)_{i=1}^{\ell} = -\nabla_z \sum_{i=1}^{\ell} L(y_i, z_i)\big|_{z_i=a_{N-1}(x_i)}$$

При таком выборе сдвигов s_i мы, по сути, сделаем один шаг градиентного спуска, двигаясь в сторону наискорейшего убывания ошибки на обучающей выборке. Отметим, что речь идет о градиентном спуске в ℓ-мерном пространстве предсказаний алгоритма на объектах обучающей выборки. Поскольку вектор сдвига будет свой на каждой итерации, правильнее обозначать его как $s_i^{(N)}$, но для простоты будем иногда опускать верхний индекс.

Итак, мы поняли, какие значения новый алгоритм должен принимать на объектах обучающей выборки. По данным значениям в конечном числе точек необходимо построить функцию, заданную на всем пространстве объектов. Это классическая задача обучения с учителем, которую мы уже хорошо умеем решать. Один из самых простых функционалов — среднеквадратичная ошибка. Воспользуемся им для поиска базового алгоритма, приближающего градиент функции потерь на обучающей выборке:

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

Отметим, что здесь мы оптимизируем квадратичную функцию потерь независимо от функционала исходной задачи — вся информация о функции потерь L находится в антиградиенте s_i, а на данном шаге лишь решается задача аппроксимации функции по ℓ точкам. Разумеется, можно использовать и другие функционалы, но среднеквадратичной ошибки, как правило, оказывается достаточно. Ещё одна причина для использования среднеквадратичной ошибки состоит в том, что от алгоритма требуется как можно точнее приблизить направление наискорейшего убывания функционала (то есть направление $(s_i)_i$); совпадение направлений вполне логично оценивать через косинус угла между ними, который напрямую связан со среднеквадратичной ошибкой.

После того, как новый базовый алгоритм найден, можно подобрать коэффициент при нем по аналогии с наискорейшим градиентным спуском:

$$\gamma_N = \arg\min_{\gamma \in \mathbb{R}} \sum_{i=1}^{\ell} L(y_i, a_{N-1}(x_i) + \gamma b_N(x_i))$$

Описанный подход с аппроксимацией антиградиента базовыми алгоритмами и называется градиентным бустингом. Данный метод представляет собой поиск лучшей функции, восстанавливающей истинную зависимость ответов от объектов, в пространстве всех возможных функций. Ищем мы данную функцию с помощью «псевдоградиентного» спуска — каждый шаг делается вдоль направления, задаваемого некоторым базовым алгоритмом. При этом сам базовый алгоритм выбирается так, чтобы как можно лучше приближать антиградиент ошибки на обучающей выборке.

6.6.3 Регуляризация

Сокращение шага. На практике оказывается, что градиентный бустинг очень быстро строит композицию, ошибка которой на обучении выходит на асимптоту, после чего начинает настраиваться на шум и переобучаться. Это явление можно объяснить одной из двух причин:

- Если базовые алгоритмы очень простые (например, решающие деревья небольшой глубины), то они плохо приближают вектор антиградиента. По сути, добавление такого базового алгоритма будет соответствовать шагу вдоль направления, сильно отличающегося от направления наискорейшего убывания. Соответственно, градиентный бустинг может свестись к случайному блужданию в пространстве.

- Если базовые алгоритмы сложные (например, глубокие решающие деревья), то они способны за несколько шагов бустинга идеально подогнаться под обучающую выборку — что, очевидно, будет являться переобучением, связанным с излишней сложностью семейства алгоритмов.

Хорошо зарекомендовавшим себя способом решения данной проблемы является *сокращение шага*: вместо перехода в оптимальную точку в направлении антиградиента делается укороченный шаг

$$a_N(x) = a_{N-1}(x) + \eta \gamma_N b_N(x),$$

где $\eta \in (0, 1]$ — темп обучения [16]. Как правило, чем меньше темп обучения, тем лучше качество итоговой композиции. Сокращение шага, по сути, позволяет понизить доверие к направлению, восстановленному базовым алгоритмом.

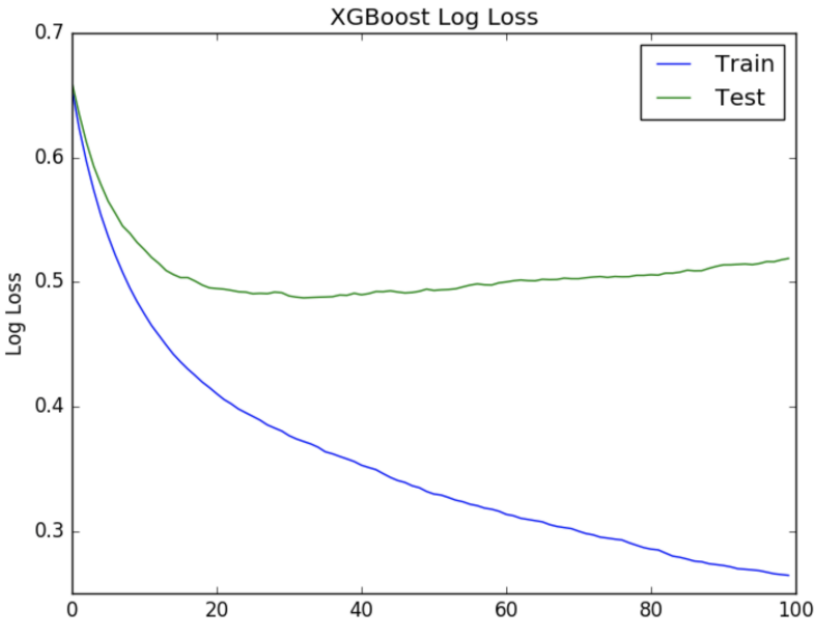

Рис. 11. Переобучение градиентного бустинга.

Также следует обратить внимание на число итераций градиентного бустинга. Хотя ошибка на обучении монотонно стремится к нулю, ошибка на контроле, как правило, начинает увеличиваться после определенной итерации. Оптимальное число итераций можно выбирать, например, по отложенной выборке или с помощью кросс-валидации.

Стохастический градиентный бустинг. Еще одним способом улучшения качества градиентного бустинга является внесение рандомизации в процесс обучения базовых алгоритмов [17]. А именно, алгоритм b_N обучается не по всей выборке X, а лишь по ее случайному подмножеству $X^k \subset X$. В этом случае понижается уровень шума в обучении, а также повышается эффективность вычислений. Существует рекомендация брать подвыборки, размер которых вдвое меньше исходной выборки.

6.6.4 Функции потерь

Регрессия. При вещественном целевом векторе, как правило, используют квадратичную функцию потерь, формулы для которой уже были приведены в разделе 6.6.1. Другой вариант — модуль отклонения $L(y, z) = |y - z|$, для которого антиградиент вычисляется по формуле

$$s_i^{(N)} = -\operatorname{sign}(a_{N-1}(x_i) - y_i).$$

Классификация. В задаче классификации с двумя классами разумным выбором является логистическая функция потерь, с которой уже сталкивались при изучении

линейных методов:

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

Задача поиска базового алгоритма с ней принимает вид

$$b_N = \arg\min_{b \in \mathcal{A}} \sum_{i=1}^{\ell} \left(b(x_i) - \frac{y_i}{1 + \exp(y_i a_{N-1}(x_i))} \right)^2.$$

Логистическая функция потерь имеет интересную особенность, связанную со взвешиванием объектов. Заметим, что ошибка на N-й итерации может быть записана как

$$\begin{aligned} Q(a_N) &= \sum_{i=1}^{\ell} \log\left(1 + \exp(-y_i a_N(x_i))\right) = \\ &= \sum_{i=1}^{\ell} \log\left(1 + \exp(-y_i a_{N-1}(x_i)) \exp(-y_i \gamma_N b_N(x_i))\right). \end{aligned}$$

Если отступ $y_i a_{N-1}(x_i)$ на i-м объекте большой положительный, то данный объект не будет вносить практически никакого вклада в ошибку, и может быть исключен из всех вычислений на текущей итерации без потерь. Таким образом, величина

$$w_i^{(N)} = \exp(-y_i a_{N-1}(x_i))$$

может служить мерой важности объекта x_i на N-й итерации градиентного бустинга.

6.6.5 Градиентный бустинг над деревьями

Считается, что градиентный бустинг над решающими деревьями — один из самых универсальных и сильных методов машинного обучения, известных на сегодняшний день. В частности, на градиентном бустинге над деревьями основан MatrixNet — алгоритм ранжирования компании Яндекс [18].

Вспомним, что решающее дерево разбивает все пространство на непересекающиеся области, в каждой из которых его ответ равен константе:

$$b_n(x) = \sum_{j=1}^{J_n} b_{nj}[x \in R_j],$$

где $j = 1, \ldots, J_n$ — индексы листьев, R_j — соответствующие области разбиения, b_{nj} — значения в листьях. Значит, на N-й итерации бустинга композиция обновляется как

$$a_N(x) = a_{N-1}(x) + \gamma_N \sum_{j=1}^{J_N} b_{Nj}[x \in R_j] = a_{N-1}(x) + \sum_{j=1}^{J_N} \gamma_N b_{Nj}[x \in R_j].$$

Видно, что добавление в композицию одного дерева с J_N листьями равносильно добавлению J_N базовых алгоритмов, представляющих собой предикаты вида $[x \in R_j]$. Если бы вместо общего коэффициента γ_N был свой коэффициент γ_{Nj} при каждом предикате, то мы могли бы его подобрать так, чтобы повысить качество композиции.

Если подбирать свой коэффициент γ_{Nj} при каждом слагаемом, то потребность в b_{Nj} отпадает, его можно просто убрать:

$$\sum_{i=1}^{\ell} L\left(y_i, a_{N-1}(x_i) + \sum_{j=1}^{J_N} \gamma_{Nj}[x \in R_j]\right) \to \min_{\{\gamma_{Nj}\}_{j=1}^{J_N}}.$$

Поскольку области разбиения R_j не пересекаются, данная задача распадается на J_N независимых подзадач:

$$\gamma_{Nj} = \arg\min_{\gamma} \sum_{x_i \in R_j} L(y_i, a_{N-1}(x_i) + \gamma), \qquad j = 1, \ldots, J_N.$$

В некоторых случаях оптимальные коэффициенты могут быть найдены аналитически — например, для квадратичной и абсолютной ошибки.

Рассмотрим теперь логистическую функцию потерь. В этом случае нужно решить задачу

$$F_j^{(N)}(\gamma) = \sum_{x_i \in R_j} \log\left(1 + \exp\left(-y_i(a_{N-1}(x_i) + \gamma)\right)\right) \to \min_{\gamma}.$$

Данная задача может быть решена лишь с помощью итерационных методов, аналитической записи для оптимального γ не существует. Однако на практике обычно нет необходимости искать точное решение — оказывается достаточным сделать лишь один шаг метода Ньютона-Рафсона из начального приближения $\gamma_{Nj} = 0$. Можно показать, что в этом случае

$$\gamma_{Nj} = \frac{\partial F_j^{(N)}(0)}{\partial \gamma} \Bigg/ \frac{\partial^2 F_j^{(N)}(0)}{\partial \gamma^2} = -\sum_{x_i \in R_j} s_i^{(N)} \Bigg/ \sum_{x_i \in R_j} |s_i^{(N)}|(1 - |s_i^{(N)}|).$$

Смещение и разброс. В случайных лесах используются глубокие деревья, поскольку от базовых алгоритмов требуется низкое смещение; разброс же устраняется за счёт усреднения ответов различных деревьев. Бустинг работает несколько иначе — в нём каждый следующий алгоритм целенаправленно понижает ошибку композиции, и даже при использовании простейших базовых моделей композиция может оказаться достаточно сложной. Более того, итоговая композиция вполне может оказаться переобученной при большом количестве базовых моделей. Это означает, что благодаря бустингу можно понизить смещение моделей, а разброс либо останется таким же, либо увеличится. Из-за этого, как правило, в бустинге используются неглубокие решающие деревья (3-6 уровней), которые обладают большим смещением, но не склонны к переобучению.

6.6.6 Взвешивание объектов

Одним из первых широко распространённых методов построения композиций является AdaBoost, в котором оптимизируется экспоненциальная функция потерь $L(y, z) = = e^{-yz}$. Благодаря её свойствам удаётся свести задачу поиска базового алгоритма к

минимизации доли неверных ответов с весами при объектах. Эти веса возникают и в градиентном бустинге при использовании экспоненциальной функции потерь:

$$L(a, X) = \sum_{i=1}^{\ell} \exp\left(-y_i \sum_{n=1}^{N} \gamma_n b_n(x_i)\right).$$

Найдем компоненты ее антиградиента после $(N-1)$-й итерации:

$$s_i = -\left.\frac{\partial L(y_i, z)}{\partial z}\right|_{z=a_{N-1}(x_i)} = y_i \underbrace{\exp\left(-y_i \sum_{n=1}^{N-1} \gamma_n b_n(x_i)\right)}_{w_i}.$$

Заметим, что антиградиент представляет собой ответ на объекте, умноженный на его вес. Если все веса будут равны единице, то следующий базовый классификатор будет просто настраиваться на исходный целевой вектор $(y_i)_{i=1}^{\ell}$; штраф за выдачу ответа, противоположного правильному, будет равен 4 (поскольку при настройке базового алгоритма используется квадратичная функция потерь). Если же какой-либо объект будет иметь большой отступ, то его вес окажется близким к нулю, и штраф за выдачу любого ответа будет равен 1.

Отметим, что многие функционалы ошибки классификации выражаются через отступы объектов:

$$L(a_{N-1}, X^{\ell}) = \sum_{i=1}^{\ell} L(a_{N-1}(x_i), y_i) = \sum_{i=1}^{\ell} \tilde{L}(y_i a_{N-1}(x_i)).$$

В этом случае антиградиент принимает вид

$$s_i = y_i \underbrace{\left(-\frac{\partial \tilde{L}(y_i a_{N-1}(x_i))}{\partial a_{N-1}(x_i)}\right)}_{w_i},$$

то есть тоже взвешивает ответы с помощью ошибки на них.

6.6.7 Влияние шума на обучение

Выше мы находили формулу для антиградиента при использовании экспоненциальной функции потерь:

$$s_i = y_i \underbrace{\exp\left(-y_i \sum_{n=1}^{N-1} \gamma_n b_n(x_i)\right)}_{w_i}.$$

Заметим, что если отступ на объекте большой и отрицательный (что обычно наблюдается на шумовых объектах), то вес становится очень большим, причем он никак не ограничен сверху. В результате базовый классификатор будет настраиваться исключительно на шумовые объекты, что может привести к неустойчивости его ответов и переобучению.

Рассмотрим теперь логистическую функцию потерь, которая также может использоваться в задачах классификации:

$$L(a, X^\ell) = \sum_{i=1}^{\ell} \log\left(1 + \exp\left(-y_i a(x_i)\right)\right).$$

Найдем ее антиградиент после $(N-1)$-го шага:

$$s_i = y_i \underbrace{\frac{1}{1 + \exp(y_i a_{N-1}(x_i))}}_{=w_i^{(N)}}.$$

Теперь веса ограничены сверху единицей. Если отступ на объекте большой отрицательный (то есть это выброс), то вес при нем будет близок к единице; если же отступ на объекте близок к нулю (то есть это объект, на котором классификация неуверенная, и нужно ее усиливать), то вес при нем будет примерно равен $1/2$. Таким образом, вес при шумовом объекте будет всего в два раза больше, чем вес при нормальных объектах, что не должно сильно повлиять на процесс обучения.

6.6.8 Методы оптимизации второго порядка

Как мы выяснили выше, градиентный бустинг осуществляет градиентный спуск в пространстве прогнозов алгоритма на обучающей выборке. Здесь может возникнуть вполне логичный вопрос: а почему бы не воспользоваться другим, более эффективным методом оптимизации? Наиболее явными кандидатами являются методы оптимизации второго порядка — например, метод Ньютона. При оптимизации числовой функции $Q(w)$ шаг в методе Ньютона осуществляется по формуле

$$w^{(n)} = w^{(n-1)} - H^{-1}(w^{(n-1)})\nabla_w Q(w^{(n-1)}),$$

где $H(w)$ — матрица вторых производных, которая также называется матрицей Гессе.

Этим же подходом можно воспользоваться и в градиентном бустинге. Нам нужно как можно сильнее уменьшить значение следующей функции путем подбора сдвигов s_i:

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

Мы уже находили вектор градиента:

$$\nabla_s Q(s) = g = \left(\left.\frac{\partial L}{\partial z}\right|_{z=a_{N-1}(x_i)}\right)_{i=1}^{\ell}$$

Заметим, что матрица вторых производных тут будет диагональной, поскольку каждая переменная s_i входит лишь в одно отдельное слагаемое:

$$H = \operatorname{diag}\left(\left.\frac{\partial^2 L}{\partial z^2}\right|_{z=a_{N-1}(x_1)}, \ldots, \left.\frac{\partial^2 L}{\partial z^2}\right|_{z=a_{N-1}(x_\ell)}\right)$$

Мы здесь пользуемся функционалом, который представляет собой сумму ошибок на всех объектах. Такое представление подходит во многих задачах, но не является максимально общим. Дело в том, что в некоторых ситуациях необходимо использовать функционалы, которые измеряют качество сортировки объектов алгоритмом — это, иными словами, функционалы качества ранжирования. Их особенность заключается как раз в том, что матрица вторых производных уже не будет диагональной.

Зная градиент и матрицу Гессе, мы можем выписать формулу для сдвигов s:

$$s = -H^{-1}g$$

Поскольку обращение матрицы является неустойчивой операцией, правильнее будет находить вектор сдвигов через систему линейных уравнений

$$Hs = -g$$

В случае с диагональной матрицей Гессе данный подход сводится к домножению каждой компоненты вектора антиградиента на некоторые коэффициенты. В общем же случае такой подход может оказаться слишком сложным, поскольку при больших размерах выборки матрица Гессе будет получаться слишком большой для эффективной работы. После того, как рассчитан вектор сдвигов, процедура будет такой же, как и раньше — обучаем алгоритм на данные сдвиги, находим коэффициент при нем, добавляем в композицию.

На похожей идее основан метод LogitBoost, который настраивает композицию с использованием логистической функции потерь, исходя из несколько иных предположений. Использование метода Ньютона приводит к тому, что базовый алгоритм настраивается на взвешенный функционал, что может затруднять обучение. Более того, формулы для весов получаются не вполне устойчивыми, и нередко в них происходит деление на очень маленькое число. Чтобы избежать этого, вводится ряд достаточно грубых эвристик.

Обратим внимание, что трюк с переподбором прогнозов в листьях базовых решающих деревьев похож на применение метода Ньютона. Допустим, мы как-то выбрали сдвиги s_i и обучили на них решающее дерево $b_n(x)$. После этого на объекте x_i обучающей выборки будет сделан сдвиг $b_n(x_i)$. Сдвиги будут одинаковыми на тех объектах, которые попали в один и тот же лист дерева. Если сделать переподбор, то сдвиги будут изменены так, чтобы как можно сильнее уменьшать исходный функционал ошибки. По сути, благодаря этому сдвиги подбираются индивидуально под группы объектов, что близко к использованию методов второго порядка с диагональной матрицей Гессе — там тоже выставляются индивидуальные коэффициенты при компонентах сдвига.

§6.7 Extreme Gradient Boosting (XGBoost)

Мы уже разобрались с двумя типами 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. Мы рассмотрим конкретную реализацию

градиентного бустинга — пакет XGBoost [19], который считается одним из лучших на сегодняшний день. Это подтверждается, например, активным его использованием в соревнованиях по анализу данных на `kaggle.com`.

6.7.1 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:

$$s = \left(-\frac{\partial L}{\partial z}\Big|_{z=a_{N-1}(x_i)} \right)_{i=1}^{\ell} = -\nabla_z \sum_{i=1}^{\ell} L(y_i, z_i)\Big|_{z_i=a_{N-1}(x_i)} \tag{6.5}$$

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

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

6.7.2 Альтернативный подход

Выше 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:

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

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

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

где через h_i обозначены вторые производные по сдвигам:

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

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

$$\sum_{i=1}^{\ell} \left(-s_i b(x_i) + \frac{1}{2} h_i b^2(x_i) \right) \to \min_b \tag{6.6}$$

Покажем, что он очень похож на среднеквадратичный из формулы (6.5). Преобразуем его:

$$\sum_{i=1}^{\ell} (b(x_i) - s_i)^2$$
$$= \sum_{i=1}^{\ell} \left(b^2(x_i) - 2s_i b(x_i) + s_i^2\right) = \{\text{последнее слагаемое не зависит от } b\}$$
$$= \sum_{i=1}^{\ell} \left(b^2(x_i) - 2s_i b(x_i)\right)$$
$$= 2\sum_{i=1}^{\ell} \left(-s_i b(x_i) + \frac{1}{2} b^2(x_i)\right) \to \min_b$$

Видно, что последняя формула совпадает с (6.6) с точностью до константы, если положить $h_i = 1$. Таким образом, в обычном градиентном бустинге мы используем аппроксимацию второго порядка при обучении очередного базового алгоритма, и при этом отбрасываем информацию о вторых производных (то есть считаем, что функция имеет одинаковую кривизну по всем направлениям).

6.7.3 Регуляризация

Будем далее работать с функционалом (6.6). Он измеряет лишь ошибку композиции после добавления нового алгоритма, никак при этом не штрафуя за излишнюю сложность этого алгоритма. Ранее мы решали проблему переобучения путем ограничения глубины деревьев, но 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 $\|b\|_2^2 = \sum_{j=1}^{J} b_j^2$. 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:

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

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

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

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:

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

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

$$H(b) = -\frac{1}{2}\sum_{j=1}^{J}\frac{S_j^2}{H_j + \lambda} + \gamma J \tag{6.7}$$

6.7.4 Обучение решающего дерева

Мы получили функционал $H(b)$, который для заданной структуры дерева вычисляет минимальное значение ошибки (6.6), 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 $[x_j < t]$ в вершине R так, чтобы оно решало следующую задачу максимизации:

$$Q = H(R) - H(R_\ell) - H(R_r) \to \max,$$

где информативность вычисляется по формуле

$$H(R) = -\frac{1}{2}\left(\sum_{(h_i,s_i)\in R} s_j\right)^2 \Bigg/ \left(\sum_{(h_i,s_i)\in R} h_j + \lambda\right) + \gamma.$$

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

6.7.5 Заключение

Итак, градиентный бустинг в XGBoost имеет ряд важных особенностей.

1. Базовый алгоритм приближает направление, посчитанное с учетом вторых производных функции потерь.

2. Функционал регуляризуется — добавляются штрафы за количество листьев и за норму коэффициентов.

3. При построении дерева используется критерий информативности, зависящий от оптимального вектора сдвига.

4. Критерий останова при обучении дерева также зависит от оптимального сдвига.

§6.8 Стекинг

Разумеется, существуют способы построения композиций помимо бустинга и бэггинга. Большую популярность имеет *стекинг*, в котором прогнозы алгоритмов объявляются новыми признаками, и поверх них обучается ещё один алгоритм (который иногда называют мета-алгоритмом). Стекинг очень популярен в соревнованиях по анализу данных, поскольку позволяет агрегировать разные модели (различные композиции, линейные модели, нейросети и т.д.; иногда в качестве базовых алгоритмов могут выступать результаты градиентного бустинга с разными значениями гиперпараметров).

Допустим, мы независимо обучили N базовых алгоритмов $b_1(x), \ldots, b_N(x)$ на выборке X, и теперь хотим обучить на их прогнозах мета-алгоритма $a(x)$. Самым простым вариантом будет обучить его на этой же выборке:

$$\sum_{i=1}^{\ell} L(y_i, a(b_1(x_i), \ldots, b_N(x_i))) \to \min_a$$

При таком подходе $a(x)$ будет отдавать предпочтение тем базовым алгоритмам, которые сильнее всех подогнались под целевую переменную на обучении (поскольку по их прогнозам лучше всего восстанавливаются истинные ответы). Если среди базовых алгоритмов будет идеально переобученный (то есть запомнивший ответы на всей обучающей выборке), то мета-алгоритму будет выгодно использовать только прогнозы данного переобученного базового алгоритма, поскольку это позволит добиться лучших результатов с точки зрения записанного функционала. При этом такой мета-алгоритм, конечно, будет показывать очень низкое качество на новых данных.

Чтобы избежать таких проблем, следует обучать базовые алгоритмы и мета-алгоритм на разных выборках. Разобьём нашу обучающую выборку на K блоков $X_1, \ldots, X_K$, и обозначим через $b_j^{-k}(x)$ базовый алгоритм $b_j(x)$, обученный по всем блокам, кроме k-го. Тогда функционал для обучения мета-алгоритма можно записать как

$$\sum_{k=1}^{K} \sum_{(x_i, y_i) \in X_k} L\left(y_i, a(b_1^{-k}(x_i), \ldots, b_N^{-k}(x_i))\right) \to \min_a$$

В данном случае при вычислении ошибки мета-алгоритма на объекте x_i используются базовые алгоритмы, которые не видели этот объект при обучении, и поэтому мета-алгоритм не может переобучиться на их прогнозах.

6.8.1 Блендинг.

Частным случаем стекинга является блендинг, в котором мета-алгоритм является линейным:

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

Это самый простой способ объединить несколько алгоритмов в композицию. Иногда даже блендинг без обучения весов (то есть вариант с $w_1 = \cdots = w_N = 1/N$) позволяет улучшить качество по сравнению с отдельными базовыми алгоритмами.

6.8.2 Категориальные и текстовые признаки.

Категориальные и текстовые признаки могут быть серьёзной помехой для использования композиций над деревьями. Стандартным способом кодирования является бинаризация для категориальных признаков и TF-IDF для текстовых, что приводит к очень большой размерности признакового пространства. Случайный лес на таком наборе признаков будет обучаться долго из-за большой глубины деревьев, а градиентный бустинг может показать слишком плохие результаты из-за небольшой глубины базовых деревьев (например, глубина 4 позволяет учитывать лишь зависимость целевой переменной от наборов из 4-х признаков; в случае с текстами ответы могут зависеть от существенно более крупных наборов слов).

Одним из решений этой проблемы может стать стекинг, в котором градиентным бустингом обучается мета-алгоритм $a(x)$, а каждый категориальный и текстовый признак схлопывается в одно число соответствующим базовым алгоритмом. Для категориальных признаков базовый алгоритм, например, может вычислять счётчики — при этом обратим внимание, что мы уже отмечали важность разделения обучающих выборок для счётчиков и настраиваемых поверх них моделей. Также популярным выбором для базовых алгоритмов являются линейные модели.

7 Обучение без учителя

§7.1 Вопросы для самопроверки

1. Опишите одномерные подходы к отбору признаков с помощью дисперсии, корреляции и t-score. Какие у них недостатки?

2. Как устроен метод главных компонент? Как он формирует новые признаки?

3. Дайте определение задачи кластеризации. Чем она отличается от задач классификации и регрессии? Опишите графовый алгоритм кластеризации, основанный на компонентах связности.

4. Задача кластеризации. Метрики качества.

5. Опишите алгоритм кластеризации K-Means. Как в нём можно выбирать количество кластеров?

6. Опишите алгоритм кластеризации DBSCAN. Какие у него есть параметры?

7. Метод K-Means, вывод его шагов.

8. Задача визуализации и метод t-SNE.

8 Матричные разложения и рекомендательные системы

1. Постановка задачи построения рекомендаций. Коллаборативные методы: memory-based подход.

2. Опишите подход user-based collaborative filtering к построению рекомендательной системы.

+ Матрично-векторное дифференцирование.
+ Какие способы калибровки вероятностей вы знаете? В чём они заключаются?

Список литературы

[1] *Robbins, H., Monro S.* (1951). A stochastic approximation method. // Annals of Mathematical Statistics, 22 (3), p. 400-407.

[2] *Schmidt, M., Le Roux, N., Bach, F.* (2013). Minimizing finite sums with the stochastic average gradient. // Arxiv.org.

[3] *Flaxman, Abraham D. and Kalai, Adam Tauman and McMahan, H. Brendan* (2005). Online Convex Optimization in the Bandit Setting: Gradient Descent Without a Gradient. // Proceedings of the Sixteenth Annual ACM-SIAM Symposium on Discrete Algorithms.

[4] *Jaderberg, M. et. al* (2016). Decoupled Neural Interfaces using Synthetic Gradients. // Arxiv.org.

[5] *Davis J., Goadrich M.* (2006). The Relationship Between Precision-Recall and ROC Curves. // Proceedings of the 23rd International Conference on Machine Learning, Pittsburgh, PA.

[6] *Mohri, M., Rostamizadeh, A., Talwalkar, A.* Foundations of Machine Learning. // MIT Press, 2012.

[7] *Мерков. А. Б.* Введение в методы статистического обучения. // http://www.recognition.mccme.ru/pub/RecognitionLab.html/slbook.pdf

[8] *Bishop, C.M.* Pattern Recognition and Machine Learning. // Springer, 2006.

[9] *Crammer, K., Singer, Y.* On the Algorithmic Implementation of Multiclass Kernel-based Vector Machines. // Journal of Machine Learning Research, 2:265-292, 2001.

[10] *Tai, Farbound and Lin, Hsuan-Tien.* Multilabel Classification with Principal Label Space Transformation. // Neural Comput., 24-9, 2012.

[11] *Дьяконов А. Г.* Методы решения задач классификации с категориальными признаками. // Прикладная математика и информатика. Труды факультета Вычислительной математики и кибернетики МГУ имени М.В. Ломоносова. 2014.

[12] *Hastie T., Tibshirani R., Friedman J.* (2009). The Elements of Statistical Learning.

[13] *Hastie, T., Tibshirani, R., Friedman, J.* (2001). The Elements of Statistical Learning. // Springer, New York.

[14] *Domingos, Pedro* (2000). A Unified Bias-Variance Decomposition and its Applications. // In Proc. 17th International Conf. on Machine Learning.

[15] *Breiman, Leo* (2001). Random Forests. // Machine Learning, 45(1), 5–32.

[16] *Friedman, Jerome H.* (2001). Greedy Function Approximation: A Gradient Boosting Machine. // Annals of Statistics, 29(5), p. 1189–1232.

[17] *Friedman, Jerome H.* (1999). Stochastic Gradient Boosting. // Computational Statistics and Data Analysis, 38, p. 367–378.

[18] *Gulin, A., Karpovich, P.* (2009). Greedy function optimization in learning to rank. http://romip.ru/russir2009/slides/yandex/lecture.pdf

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