

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РФ
НОВОСИБИРСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
Механико-математический факультет

В. М. Неделько

Машинное обучение в вероятностной постановке

Курс лекций. Версия от 21.10.2024

Это черновая версия. Возможны опечатки и ошибки. Некоторые разделы написаны схематично. Многие места стилистически не проработаны.

Новосибирск
2024

УДК 517.52
ББК 22.161.3
А345

Рецензент:

...,

Неделько, В. М.

А345 Машинное обучение в вероятностной постановке : учебник / В. М. Неделько ; Новосиб. гос. ун-т. – Новосибирск : ИПЦ НГУ, 2024. – 280 с.

ISBN 978-5-4437-присваивает ИПЦ

Курс знакомит с научным направлением, известным как интеллектуальный анализ данных, а также машинное обучение, data mining, распознавание образов, теория статистических решений. Основной задачей данной научной области является разработка и обоснование методов автоматического поиска закономерностей в данных для получения новых знаний и прогнозирования.

Представлен базовый курс по интеллектуальному анализу данных. Будут рассмотрены как классические методы дискриминантного и регрессионного анализа, так и современные методы машинного обучения, в частности, нейронные сети, логистическая регрессия, SVM, деревья решений, композиции (в т.ч. boosting). Особое внимание будет уделено установлению взаимосвязи разных методов, вопросам выбора подходящего метода в конкретной ситуации, а также управлению процессом обучения и оцениванию качества построенных решений.

Материал предназначен для студентов, интересующихся данным направлением, а также специалистов в области анализа данных.

УДК 517.52
ББК 22.161.3

Рекомендовано к печати кафедрой Теоретической кибернетики
Механико-математического факультета НГУ (протокол № ... от ...)

ISBN 978-5-4437-xxxx-x © Новосибирский государственный
университет, 2024

Оглавление

Введение	9
1 Метод прецедентов	15
1.1 Метод ближайшего соседа в задаче классификации	16
1.2 Метод ближайшего соседа для вещественной целевой переменной	19
1.3 Характеристика метода прецедентов	23
1.3.1 Варианты метода прецедентов	23
1.3.2 Свойства метода прецедентов	23
1.4 Примеры кода	24
2 (Квази)линейные методы классификации	29
2.1 Решающая функция для нормальных распределений	29
2.1.1 Одномерный случай	30
2.1.2 Многомерный случай	33
2.1.3 Оценивание параметров	35
2.1.4 Фильтрация нетипичных значений	37
2.1.5 Свойства метода	38
2.2 Дискриминант Фишера	39
2.2.1 Идея метода	39
2.2.2 Свойства метода	41
2.3 Логистическая регрессия	41
2.3.1 Идея метода	41
2.3.2 Оценивание параметров	43
2.3.3 Функция потерь	43
2.3.4 Свойства метода	44

2.4	Наивный байесовский классификатор	45
2.4.1	Случай двух классов	45
2.4.2	Случай нескольких классов	47
2.4.3	Ряд Бахадура	49
2.4.4	Свойства метода	51
2.5	Метод опорных векторов	52
2.5.1	Постановка задачи классификации	53
2.5.2	Линейно разделимая выборка	54
2.5.3	Линейно неразделимая выборка	56
2.5.4	Двойственная задача	57
2.5.5	Ядра и спрямляющие пространства	60
2.5.6	Беспризнаковая классификация	62
2.5.7	Задача безусловной оптимизации	63
2.5.8	Свойства метода	64
2.6	Характеристика линейных методов	65
2.7	Примеры кода	66
2.7.1	Визуализация решений	66
2.7.2	Кодирование категориальных переменных и наивный байесовский классификатор	69
3	Логические методы классификации	71
3.1	Логические закономерности	71
3.1.1	Понятие закономерности	71
3.1.2	Критерии качества закономерности	74
3.1.3	Решающие списки	77
3.2	Решающие деревья	77
3.2.1	Понятие дерева решений	78
3.2.2	Построение дерева решений	78
3.2.3	Критерии ветвления	82
3.2.4	Алгоритмы	84
3.3	Выводы	84
3.4	Примеры кода	85
4	Ансамблевые методы	87
4.1	Композиции решающих функций	87
4.2	Независимые подвыборки	88
4.2.1	Бэггинг	89

4.2.2	Случайный лес	90
4.2.3	Свойства	91
4.3	Бустинг	91
4.3.1	Алгоритм AdaBoost	92
4.3.2	Свойства	94
4.4	Нейронные сети	95
4.4.1	Схема нейрона	95
4.4.2	Аппроксимирующая способность сети	96
4.4.3	Обучение сети	96
4.4.4	Архитектуры нейронных сетей	99
4.4.5	Глубокое обучение	100
4.4.6	Бустинг и нейронные сети	100
4.4.7	Свойства нейронных сетей	102
4.5	Ансамбли на основе различных методов	103
4.5.1	Простое голосование	103
4.5.2	Стэкинг	104
4.6	Выводы	105
4.6.1	Бустинг и случайный лес	107
4.7	Примеры кода	107
4.7.1	Подбор сложности ансамбля	107
4.7.2	Визуализация нейронной сети	113
5	Градиентный бустинг	116
5.1	Алгоритм AdaBoost	116
5.1.1	Непосредственная оптимизация экспоненциальной функции потерь	117
5.1.2	AdaBoost как градиентный бустинг	119
5.1.3	Сходимость процесса бустинга	121
5.2	Градиентный бустинг	122
5.3	Метод XGBoost	125
5.4	Свойства бустинга	130
5.5	Выводы	134
5.6	Примеры кода	136
6	Оценивание качества решений	137
6.1	Постановка задачи	137
6.2	Контрольная выборка	141

6.2.1	Точечная оценка	142
6.2.2	Доверительный интервал	142
6.2.3	Усреднённые оценки	145
6.2.4	Область применимости байесовского подхода	146
6.2.5	Случай произвольной функции потерь	147
6.2.6	Свойства оценки по контрольной выборке .	152
6.3	Эмпирический риск	153
6.3.1	Смещённость эмпирического риска	153
6.3.2	Оценки Вапника-Червоненкиса	155
6.3.3	Функция роста	157
6.3.4	Неулучшаемость оценок	158
6.3.5	Точные асимптотические оценки для гистограммного классификатора	163
6.3.6	Свойства эмпирического риска	164
6.4	Скользящий экзамен	166
6.4.1	Определение	166
6.4.2	Несмешённость скользящего экзамена	168
6.4.3	Некорректное оценивание дисперсии	169
6.4.4	Точность скользящего экзамена	169
6.4.5	Разновидности скользящего экзамена	172
6.4.6	Оценка out-of-bag	175
6.4.7	Свойства оценки скользящего экзамена . . .	176
6.5	Доверительный интервал для риска	177
6.5.1	Качество точечных оценок	177
6.5.2	Доверительный интервал для риска	179
6.6	Выводы	182
6.7	Примеры кода	182
7	Задача машинного обучения	187
7.1	Критерии качества	187
7.1.1	Целевая функция потерь	187
7.1.2	Критерии точности классификации	189
7.1.3	Функция потерь для оценки вероятности .	191
7.1.4	Функция потерь для непрерывной целевой переменной	191
7.2	Разложение ошибки	191
7.2.1	Bias-variance decomposition	192

7.2.2	Теория статистической устойчивости	194
7.2.3	Процесс обучения	194
7.2.4	Выводы	195
7.3	Общая постановка задачи машинного обучения	196
7.3.1	Модели	196
7.3.2	Задача построения решающей функции	197
7.3.3	Связь с задачей проверки гипотез	198
7.3.4	Байесовский подход	198
7.3.5	Случайность и Колмогоровская сложность .	199
7.3.6	Классификация методов машинного обучения	202
7.4	Сравнение методов построения решающих функций	203
7.4.1	Выводы	206
7.5	Некоторые задачи машинного обучения	207
8	Некоторые задачи анализа данных	209
8.1	Методы глобальной оптимизации	209
8.1.1	Задача поиска глобального экстремума	210
8.1.2	Формальная постановка	211
8.1.3	Теорема о небесплатном завтраке	212
8.1.4	Алгоритм СПА	214
8.2	Кластерный анализ	215
8.2.1	Метод k -средних	215
8.2.2	Оценивание смеси распределений	216
8.2.3	Иерархическая кластеризация	216
8.3	Анализ временных рядов	217
8.4	Прогнозирование нескольких переменных	219
8.5	Использование экспертных знаний	219
8.5.1	Постановка задачи	220
8.5.2	Байесовский подход	221
8.5.3	Статистическая интерпретация экспертной информации	222
8.5.4	Примеры	222
9	Проблемы искусственного интеллекта	224
9.1	Большие языковые модели	225
9.1.1	Тестирование возможностей	225
9.1.2	Перспективы использования	235

9.1.3	Перспективы развития	238
9.2	Историческая ретроспектива ИИ	240
9.2.1	Краткий обзор	240
9.2.2	Экспертные системы	242
9.3	Естественный интеллект	245
9.3.1	Интеллект и сознание	245
9.3.2	Виды познания	247
9.4	Примеры кода	248
	Заключение	251

Введение

В эпоху, когда машинным обучением не занимается только ленивый... Пожалуй, такая формулировка не будет соответствовать реальности. В реальности многие инструменты машинного обучения ориентированы как раз на «ленивого» пользователя: это и полностью сконфигурированные предварительно обученные нейросети, и инструменты для автоматического подбора параметров, и даже инструменты для автоматического выбора метода обучения (модели). Поэтому точнее будет сформулировать тезис следующим образом. В условиях, когда машинным обучением занимается даже ленивый, нет необходимости обосновывать актуальность этого научного направления, поскольку она очевидна.

Возможно, текущие ожидания в этой области даже несколько (а может быть и значительно) завышены. Однако реальная польза данных инструментов не вызывает сомнений, равно как и их востребованность в обозримом будущем.

Следует уточнить терминологию. Мы будем использовать как синонимы следующие термины: машинное обучение, интеллектуальный анализ данных, теория статистических решений, data mining, распознавание образов.

Также будем использовать широкий термин «искусственный интеллект», относя к этой области любые вопросы, связанные с автоматизацией интеллектуальной деятельности, т.е. с передачей компьютеру интеллектуальных функций человека.

Представленный материал в первую очередь относится к машинному обучению, но некоторые вопросы будут касаться как раз более широкой области.

Можно выделить два подхода к использованию машинного обучения, которые условно назовём «инженерным» и «исследовательским».

Первый подход предполагает использование готовых «рецептов» для построения моделей машинного обучения. Для этого требуется широкий кругозор, изучение последних актуальных инструментов и знание большого числа шаблонов для решения разных практических задач. При этом не требуется глубокого понимания используемых методов и приёмов. По большому счёту достичь такого понимания для всех актуальных инструментов физически нереально ввиду их большого количества и постоянного обновления. Подобная квалификация широко востребована. Вместе с тем, такие специалисты испытывают затруднения при решении нестандартных задач, требующих доработки или модификации известных методов.

Чтобы решать нестандартные задачи или разрабатывать новые методы анализа данных, требуется хорошо разобраться в их принципах и тонкостях работы. При этом не получится ограничиться только одной областью машинного обучения (например, только нейросетевыми методами). Нейронные сети являются основным инструментом при решении многих задач анализа данных (хотя и далеко не всех, например, на табличных данных нейросети, как правило, уступают бустингу), но инструментом, сложным для анализа и неподходящим для изучения принципов машинного обучения. Чтобы научиться понимать сложные нейросетевые модели, целесообразно потренироваться на более простых моделях.

Поэтому если ставить целью получение не только инженерной, но и исследовательской квалификации, то целесообразно системно изучать основные методы и подходы машинного обучения, от простых к более сложным. Именно такую цель преследует данное учебное пособие. Большое внимание будет уделено выявлению неочевидных связей между различными методами, а также фундаментальным проблемам машинного обучения. Необходимо разобраться с математическим фундаментом, на котором основывается область искусственного интеллекта.

Следует отметить отличия данного учебника от классических

курсов (например, [1, 2]), а также большого количества открытых курсов и пособий по машинному обучению.

Во-первых, существенно отличается структура и изложение материала. Это связано с тем, что в анализе данных существуют разные научные школы, и если математические факты для всех одинаковы, то их интерпретация и практика применения в разных школах могут различаться. Разумеется, автор не считает «своё» изложение более «правильным» по сравнению с другими. Материал излагается так, как было бы понятнее самому автору. Предполагается, что читателю в любом случае, полезно рассмотреть материал с разных точек зрения.

Во-вторых, здесь, несмотря на относительно небольшой объём, есть материал, который редко встречается в других источниках.

В-третьих, пособие предполагает широкую целевую аудиторию и предназначено как для начального изучения предмета, так и для специалистов по анализу данных. Такая универсальность достигается за счёт того, что темы выбираются не по их сложности, а по важности.

Рассматриваются разные вопросы, среди которых есть как простые, так и сложные. При этом для сложных вопросов предварительно даётся некоторое краткое изложение основных идей, так чтобы при первом чтении можно было пропустить сложные детали и это не помешало пониманию дальнейшего материала.

Автор будет рад получить комментарии и замечания (на адрес: nedelko@math.nsc.ru).

На первый взгляд, в математике не может быть разных точек зрения, поскольку если теорема доказана, то уже не получится с ней не согласиться. Но разные точки зрения могут быть в интерпретации утверждений. Например, теорема Гёделя о неполноте. Иногда её интерпретируют как довод в пользу того, что в математике существуют утверждения, которые нельзя ни доказать, ни опровергнуть. Но из самой теоремы подобного не следует, это всего лишь интуитивный довод на основе аналогии.

В машинном обучении очень много подобных примеров, когда некоторые строгие утверждения используются для аргументации некоторых гипотез, которые лишь косвенно связаны с доказанными

ми утверждениями.

В данном материале мы будем особо акцентировать внимание на таких «общезвестных фактах», которые на самом деле не доказаны и, вполне вероятно, фактами не являются.

Особо отметим, что это пособие не является монографией. Если в монографии результат приводится без ссылок, то это значит, что результат принадлежит автору. Если в учебнике результат приводится без ссылки — значит автор считает результат общезвестным.

В этом пособии именно так: все собственные результаты автора приведены со ссылкой; всё, что дано без ссылки — автор считает общезвестным или слишком простым, чтобы устанавливать авторство.

Автор настоятельно рекомендует (до или после чтения данного пособия) ознакомиться с [1]. Это классический курс по машинному обучению. Включает конспекты лекций, презентации и видеозаписи лекций. Кроме того, на ресурсе machinelearning имеется много полезных статей данного автора, в том числе рекомендации по ведению научной работы для студентов.

Также рекомендуется (попытаться) изучить учебник [2]. Но это только в качестве следующего шага, т.к. он труднее для понимания.

Для дальнейшего чтения можно порекомендовать также [3]. Статьи этого автора (который длительное время имел самый высокий рейтинг на kaggle) полезны как источник практических рекомендаций и приёмов для решения реальных задач.

При этом автор настоятельно не рекомендует начинать изучение предмета с online-учебников и интернет-курсов. Они зачастую представляют собой набор статей разных авторов, поэтому по ним трудно получить цельное представление о материале. Также в подобных источниках многие вопросы освещаются поверхностно.

В текущих реалиях невозможно представить себе эффективное обучение чему-либо без использования интернет-ресурсов.

Для изучения предмета автор рекомендует использовать следующие источники.

<http://kaggle.com/> — ресурс, который нужно использовать в обя-

зательном порядке. Первоначально это была просто платформа для проведения конкурсов по анализу данных. Однако очень быстро стал понятен её образовательный потенциал, а также значение как «полигона» для сравнения различных методов анализа данных, который позволяет оценить их реальную полезность. Главную образовательную ценность на этом ресурсе имеют законченные примеры решения реальных задач лучшими специалистами, представленные в виде полностью рабочего кода с пояснениями.

<http://archive.ics.uci.edu/ml/datasets.html> — UCI (University of California, Irvine) Machine Learning Repository. Наиболее известная (до появления kaggle) коллекция задач машинного обучения, содержащая несколько сотен наборов данных из различных прикладных задач. На задачи этого репозитория принято ссылаться в академических публикациях, где они используются в качестве реальных данных. Использование одинаковых данных разными исследователями делает получаемые результаты воспроизводимыми.

<http://www.machinelearning.ru/> — информационный русскоязычный ресурс по анализу данных. Содержит статьи разного уровня по качеству (важно смотреть на авторство статьи), но большое число статей полезно в качестве источника справочной информации.

<https://education.yandex.ru/handbook/ml> — достаточно качественный справочник по машинному обучению. Категорически не рекомендуется для первоначального знакомства с предметом, однако полезен для дополнительного обучения.

В названии курса говорится о машинном обучении в вероятностной постановке. Нужно оговорить, что такое название вовсе не означает каких-либо ограничений на круг рассматриваемых вопросов и методов. Мы будем рассматривать в том числе и невероятностные методы и подходы. А вероятностная постановка упомянута скорее как намерение по возможности использовать вероятностную интерпретацию задач и результатов. Иными словами,

речь будет идти о «самом обычном» машинном обучении.

Глава 1

Метод прецедентов

Метод прецедентов является, пожалуй, наиболее простым методом машинного обучения и наиболее интуитивным. Фактически он отражает естественную человеческую стратегию поведения: для принятия решения в новой ситуации найти похожие случаи из прошлого опыта и действовать по аналогии.

Метод прецедентов является на самом деле не конкретным методом, а некоторым подходом (классом методов), в рамках которого существует ряд методов, по-разному реализующих идею прецедентов или аналогии.

Для объекта, который нужно классифицировать, находятся наиболее похожие на него объекты, для которых целевой признак известен.

Прогноз осуществляется на основе «голосования» по прецедентам.

Данный класс методов также называют метрическими методами построения решающих функций. При этом методы вовсе не требует, чтобы функция (не)похожести объектов была именно метрикой.

1.1 Метод ближайшего соседа в задаче классификации

Для знакомства с методом прецедентов рассмотрим одну простую практическую задачу машинного обучения, а именно задачу «Iris» из репозитория UCI.

Задача состоит в том, чтобы построить правило, позволяющее по внешним признакам различать три вида (сорта) ирисов (цветы):

1 – Iris Setosa,

2 – Iris Versicolour,

3 – Iris Virginica.

Вид ириса обозначим переменной Y , которая будет принимать значения 1, 2, 3.

В качестве измеряемых характеристик используются:

X_1 – длина чашелистика (sepal length),

X_2 – длина лепестка (petal length).

Таким образом, требуется по измеренным значениям переменных X_1 и X_2 определить вид ириса.

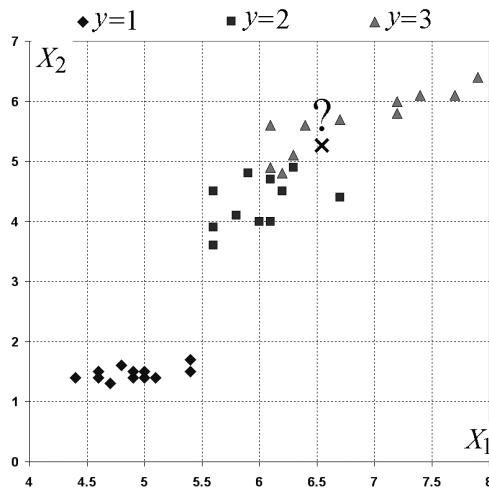
Результаты измерений указанных характеристик для 36 экземпляров растений трёх видов приведены в таблице 1.1. Здесь i обозначает номер объекта (экземпляра) и в верхней позиции обозначает не степень, а верхний индекс, x_1^i – значения переменной X_1 , x_2^i – значения переменной X_2 , y^i – значения переменной Y .

Заметим, что в исходной таблице данных, доступной в репозитории, используются четыре характеристики, а число объектов составляет 150, однако, для наглядности мы рассматриваем сокращённую таблицу.

Для наглядности изобразим имеющиеся данные на диаграмме (см. рис. 1.2). Требуется, проанализировав представленные данные (будем называть их выборкой), научиться распознавать сорт

i	x_1^i	x_2^i	y^i	i	x_1^i	x_2^i	y^i
1	5,1	1,4	1	13	6	4	2
2	4,9	1,4	1	14	6,1	4,7	2
3	4,7	1,3	1	15	5,6	3,6	2
4	4,6	1,5	1	25	6,7	5,7	3
5	5	1,4	1	26	7,2	6	3
6	5,4	1,7	1	27	6,2	4,8	3
7	4,6	1,4	1	25	6,7	5,7	3

Рис. 1.1: Данные по ирисам.



ириса по внешним признакам (X_1 и X_2). Математически это означает, что необходимо построить функцию f , сопоставляющую любой паре (x_1, x_2) значений переменных X_1, X_2 некоторое значение y переменной Y .

Такая функция называется решающей функцией или решающим правилом:

$$f : X_1 \times X_2 \rightarrow Y$$

или $y = f(x_1, x_2)$, $y \in Y$, $x_1 \in X_1$, $x_2 \in X_2$. Здесь и далее мы обозначаем одинаково переменную и множество значений, которые она принимает.

Идея метода прецедентов заключается в следующем. Для объекта, который нужно классифицировать, находятся наиболее похожие на него объекты, для которых целевой признак известен.

Простейшая модификация метода заключается в том, что распознаваемый объект относится к тому же классу, к которому принадлежит объект, на который он более всего похож. Такой метод носит название метода ближайшего соседа.

В качестве «меры (не)похожести» можно использовать расстояние ρ между точками, представляющими объекты в признаковом пространстве. Так например для точки $(6,6,5,2)$, которая обозначена на рис. 1.2 крестиком, ближайшей точкой выборки будет точка $(6,4,5,6)$, которая принадлежит третьему классу ($y = 3$), поэтому точке $(6,6,5,2)$ также припишем третий класс, т.е. $f(6,6,5,2) = 3$.

Введём формальное обозначение для выборки

$$S = ((x_1^i, x_2^i, y^i) | i = 1, \dots, N) = ((x^i, y^i) | i = 1, \dots, N),$$

где $x^i = (x_1^i, x_2^i)$, N – количество объектов (объём выборки).

Здесь мы просто ввели буквенные обозначения для таблицы данных и её содержимого.

Теперь решающую функцию, получаемую методом ближайшего соседа, можно записать формально

$$f(x) = y^{i_x}, \quad i_x = \arg \min_i \rho(x, x^i),$$

где $\arg \min$ – аргумент минимума, т.е. в данном случае это индекс i , при котором достигается минимум расстояния.

Существует модификация метода, когда вместо одной точки находятся несколько ближайших точек выборки и приписывается класс, к которому принадлежит большинство из них. Это метод k ближайших соседей (или kNN).

1.2 Метод ближайшего соседа для вещественной целевой переменной

В предыдущем разделе переменная Y принимала конечный набор значений, на которых не был определён порядок (так как на сортах ирисов не определено отношение «больше-меньше»). В таких случаях задача построения решающей функции называется задачей распознавания образов или задачей классификации (с «учителем»).

Если целевая переменная Y является вещественной, то говорят о задаче восстановления зависимостей. Задачу построения решающей функции при вещественной Y можно также называть задачей регрессионного анализа, хотя это будет не совсем точно, поскольку регрессионный анализ предполагает, что мы оцениваем функцию регрессии, т.е. функцию условного математического ожидания. Однако, функция регрессии является оптимальным решением, если в качестве критерия используется средний квадрат отклонения (MSE). Если, например, критерием является средний модуль отклонения (MAE), то оптимальным решением будет не регрессия, а условная медиана.

Кроме того, классическая постановка регрессионного анализа подразумевает, что значения переменных, кроме целевой, не являются случайными. Мы же полагаем все переменные случайными величинами.

Однако, термин «восстановление зависимостей» тоже не является точным, поскольку подразумевает, что есть некоторая «истинная» зависимость, которую нужно восстановить. Вместе с тем, в реальных задачах чаще всего никакой «объективной» зависимости нет, есть только статистическая (вероятностная) связь (зависимость) между переменными. А функциональную зависимость (решающую функцию) мы не восстанавливаем, а конструируем.

Поэтому будем пользоваться термином «построение решающей функции».

В качестве примера задачи построения решающей функции для вещественной (непрерывной) целевой переменной рассмотрим задачу «Wine Quality» из репозитория UCI.

База данных для этой задачи описывает несколько тысяч образцов вин, которые характеризуются одиннадцатью переменными, соответствующими измерениям различных химических показателей. Целевая переменная Y принимает целые значения от 3 до 8 и является экспертизой оценкой качества вина.

Поскольку дробные оценки качества не противоречат содержательному смыслу задачи, будем считать переменную Y вещественной.

Как и в предыдущем случае, уменьшим размерность задачи. Выберем две переменных:

X_1 – концентрация летучих кислот (volatile acidity),

X_2 – концентрация лимонной кислоты (citric acid).

Из всего множества объектов выберем по пять образцов для каждого значения качества. Заметим, что после такого сокращения данных решение, которое будет получено, нельзя переносить на исходную задачу, фактически мы сформулировали новую учебную задачу на основе данных UCI.

Сформированная выборка приведена в таблице на рис. 1.3 и визуализирована на рис. 1.4, где цифры рядом с точками отражают значения целевой переменной.

Задача заключается в построении решающей функции, которая по значениям химических показателей позволяет оценить качество вина.

Пусть, например, мы получили образец, для которого $x_1 = 0,5$, $x_2 = 0,35$. Он изображён точкой A на рис. 1.4. Что можно сказать о его качестве? Наиболее похожим из выборки является объект $i = 2$ со значениями $(0,44, 0,42)$, для которого $y = 3$. Таким образом, по методу ближайшего соседа оценка качества для объекта A также составит 3.

i	x_1^i	x_2^i	y^i	i	x_1^i	x_2^i	y^i	i	x_1^i	x_2^i	y^i
1	1,02	0,02	3	11	0,7	0,13	5	21	0,36	0,34	7
2	0,44	0,42	3	12	0,59	0,01	5	22	0,3	0,41	7
3	0,61	0,49	3	13	0,49	0,49	5	23	0,59	0	7
4	0,76	0,02	3	14	0,82	0,29	5	24	0,34	0,4	7
5	1,19	0	3	15	0,69	0,49	5	25	0,59	0,06	7
6	0,33	0,32	4	16	0,58	0,56	6	26	0,35	0,53	8
7	0,58	0	4	17	0,56	0,24	6	27	0,62	0,67	8
8	0,92	0,27	4	18	0,39	0,47	6	28	0,49	0,03	8
9	0,48	0,2	4	19	0,49	0,1	6	29	0,32	0,45	8
10	1,13	0,09	4	20	0,51	0,13	6	30	0,3	0,56	8

Рис. 1.3: Данные по задаче оценивания качества вин.

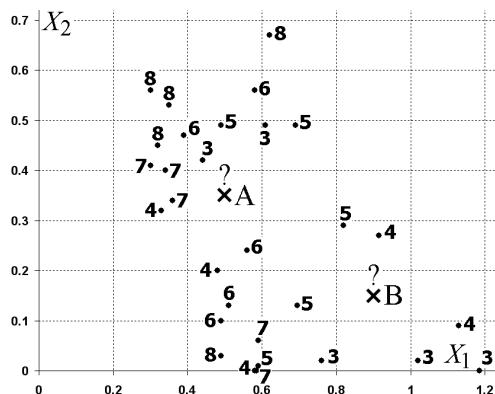


Рис. 1.4: Визуализация данных из таблицы 1.3.

Однако заметим, что недалеко от A находятся и точки, для которых $y = 7$. И при незначительном смещении точки A оценка по методу ближайшего соседа может сильно измениться.

Подобное скачкообразное изменение решающей функции представляется неоправданным, поскольку разумно считать, что качество продукта в данном случае является непрерывной функцией характеристик.

Для построения гладкой решающей функции применим так называемый ядерный (kernel-based) метод, также известный как метод потенциальных функций.

В этом методе решающая функция строится в виде

$$f(x) = \frac{1}{w} \sum_{i=1}^N y^i w_i, \quad w = \sum_{i=1}^N y^i w_i,$$

где $w_i = \varphi(\rho(x, x^i))$, $\rho(x, x^i)$ – расстояние между заданной точкой и объектом выборки, а $\varphi(\cdot)$ – потенциальная или ядерная функция.

В качестве потенциальной функции можно использовать любую монотонно убывающую функцию, например, ядро Коши

$$\varphi(z) = \frac{1}{1 + (\frac{z}{r})^2},$$

где r – параметр, задающий скорость убывания функции.

На содержательном уровне идея метода в том, что решение в каждой точке есть взвешенное среднее по выборочным значениям, причём, чем ближе выборочная точка к данной, тем в большим весом она учитывается.

Для примера вычислим оценки качества в точках A и B . При $r = 0,1$ в точке A имеем $f(0,5, 0,35) \approx 5,49$, для точки B получаем $f(0,9, 0,15) \approx 4,7$.

Данный результат выглядит достаточно разумным. Действительно, согласно выборке, качество вина выше при меньшей концентрации летучих кислот (в т.ч. уксусной), и большей концентрации лимонной кислоты.

1.3 Характеристика метода прецедентов

1.3.1 Варианты метода прецедентов

Можно выделить следующие основные разновидности метода прецедентов:

- k -ближайших соседей (прогноз на основе объектов, ближайших к исследуемому),
- парзеновское окно (прогноз на основе объектов, попавших в окрестность исследуемого объекта),
- яdroвое сглаживание (вклад объекта в прогноз есть некоторая функция расстояния).

В методе парзеновского окна фиксируется не число соседей, а форма и размер «окна», т.е. окрестности, из которой берутся соседи.

Сглаживание с использованием ядер есть наиболее общий вариант метода прецедентов. При этом функция ядра может быть как функция непосредственно расстояния, так и функция ранга по степени близости (т.е. функция номера «соседа» в порядке близости). Тогда методы kNN и парзеновского окна соответствуют ядру в виде пороговой функции.

1.3.2 Свойства метода прецедентов

Метод прецедентов, несмотря на естественность и очевидность, на практике работает относительно плохо, хотя есть исключения.

Недостатки метода:

- равное использование всех переменных, неустойчивость к «шумовым переменным», особенно при большой размерности,
- неиспользование возможной независимости переменных,
- неиспользование априорных знаний.

Основным недостатком метода является его малая пригодность для пространств большой размерности. При большом числе переменных понятие близости между точками становится малосодер-жательным и не отражает интуицию.

Представим, что в некоторой задаче есть переменная, по которой два класса безошибочно разделяются. И есть ещё 99 переменных, на которых распределения обоих классов почти совпадают (такие переменные называют «шумовыми»). В такой задаче kNN даст очень низкую точность, поскольку на расстояние между объектами влияют все переменные, и объекты, близкие по информативной переменной, весьма вероятно будут далёкими во всём пространстве.

Существуют методы, существенно более эффективные (на большинстве практических задач).

Тем не менее, метод kNN вполне можно использовать в некоторых случаях.

Во-первых, это задачи небольшой размерности, где расстояние «работает».

Во-вторых, существуют методы отбора информативных переменных, которые иногда позволяют уменьшить размерность задачи.

В-третьих, метод kNN иногда используют на случайных подпространствах небольшой размерности, чтобы сформировать новые признаки в методе стэкинга (позже его обсудим).

И в-четвёртых, существует важный подход, называемый metric learning. В этом подходе нейросеть обучается специально таким образом, чтобы сформировать признаковое описание (embedding) объекта именно так, чтобы объекты одного класса (или близкие по целевой переменной) были близкими в этом пространстве. Фактически само пространство строится именно так, чтобы kNN работал, и он в этом случае работает.

1.4 Примеры кода

Для наглядности здесь приведены примеры кода, позволяю-щего визуализировать данные, упомянутые в главе.

```

# Загружаем библиотеки:
import numpy as np
import pandas as pd
from sklearn import datasets
import seaborn as sns

# Загружаем набор данных Ирисы:
iris = datasets.load_iris()

# Формируем таблицу:
df = pd.DataFrame(iris[ 'data' ] ,
                   columns = iris[ 'feature_names' ])

# Добавляем столбец с целевой переменной:
df[ 'target' ] = iris[ 'target' ]

# Заменяем номера классов названиями сортов:
df[ 'target' ] = df[ 'target' ].map(lambda x :
                                      iris[ 'target_names' ][x])

# Смотрим, что получилось:
df

```

Результат выполнения кода приведён на рис. 1.5. Здесь видим полную таблицу данных Iris. Дополнительно показан результат операции вычисления моментов по классам.

Код запускался в среде jupyter-notebook, которая предоставляет запускать код частями и видеть результат выполнения каждого фрагмента. Также важной особенностью является то, что после модификации кода нет необходимости запускать программу с начала — можно продолжить выполнение в текущем контексте.

Пожалуй, для первоначального знакомства с языком python данная среда подходит лучше всего.

На рис. 1.6 приведён результат выполнения кода

```
sns . pairplot ( df , hue = ' target ' )
```

Показан только фрагмент рисунка, который в полном виде представляет собой 4×4 таблицу картинок и имеет слишком большой размер.

Видим, что библиотека seaborn позволяет одним вызовом получать достаточно сложные и информативные иллюстрации.

```
df
```

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	target
0	5.1	3.5	1.4	0.2	setosa
1	4.9	3.0	1.4	0.2	setosa
2	4.7	3.2	1.3	0.2	setosa
3	4.6	3.1	1.5	0.2	setosa
4	5.0	3.6	1.4	0.2	setosa
...
145	6.7	3.0	5.2	2.3	virginica
146	6.3	2.5	5.0	1.9	virginica
147	6.5	3.0	5.2	2.0	virginica
148	6.2	3.4	5.4	2.3	virginica
149	5.9	3.0	5.1	1.8	virginica

150 rows × 5 columns

```
df.groupby(['target']).agg(['mean', 'std'])
```

target	sepal length (cm)		sepal width (cm)		petal length (cm)		petal width (cm)	
	mean	std	mean	std	mean	std	mean	std
	setosa	5.006	0.352490	3.428	0.379064	1.462	0.173664	0.246
versicolor	5.936	0.516171	2.770	0.313798	4.260	0.469911	1.326	0.197753
virginica	6.588	0.635880	2.974	0.322497	5.552	0.551895	2.026	0.274650

Рис. 1.5: Выполнение кода в jupyter-notebook.

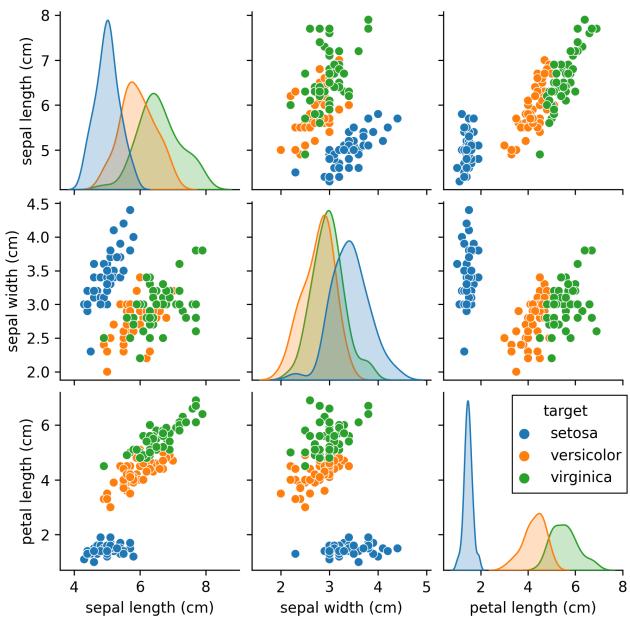


Рис. 1.6: Визуализация средствами seaborn.

В данном пособии примеров кода будет относительно мало, поскольку большой объём кода был бы неуместен. При этом примеры отобраны только наиболее важные с точки зрения иллюстрации работы методов или демонстрации важных конструкций языка.

Автор настоятельно рекомендует запустить все предложенные примеры и понять работу кода, в том числе внося в него изменения и наблюдая результат.

Глава 2

(Квази)линейные методы классификации

В данной главе рассмотрены методы классификации, в которых решающая функция является линейной, либо становится линейной в подходящем пространстве переменных.

Список методов:

- линейный и квадратичный дискриминант,
- дискриминант Фишера,
- логистическая регрессия,
- наивный байесовский классификатор,
- машина опорных векторов.

2.1 Решающая функция для нормальных распределений

Нормальное распределение является, пожалуй, наиболее часто используемым в теории вероятностей и математической статистике. В задачах анализа данных зачастую нет оснований предполагать нормальность распределений, однако, в ряде случаев такую модель оправдано использовать.

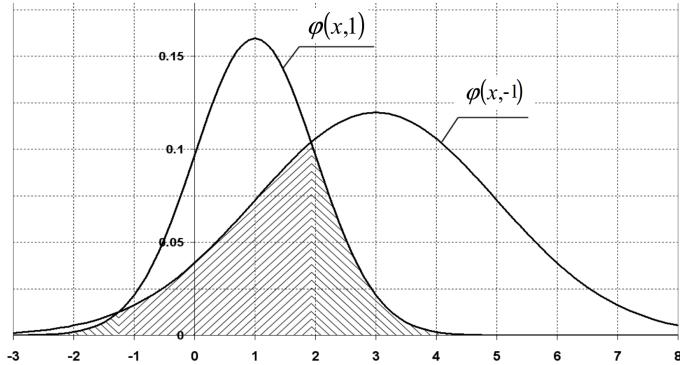


Рис. 2.1: Совместная плотность вероятности.

Рассмотрим, как выглядит оптимальное решение, если распределения известны.

2.1.1 Одномерный случай

Для простоты будем рассматривать случай двух классов, т.е. $Y = \{-1, 1\}$, и пусть имеется одна непрерывная переменная X .

Предположим, что заданы распределение $P(y) = P_y$ и условные плотности вероятности $\varphi_y(x)$, $y \in \{-1, 1\}$, причём плотности являются нормальными

$$\varphi_y(x) = \frac{1}{\sqrt{2\pi}\sigma_y} e^{-\frac{(x-\mu_y)^2}{2(\sigma_y)^2}}.$$

Через заданные величины можно выразить совместную плотность вероятности

$$\varphi(x, y) = \varphi_y(x)P(y).$$

В качестве примера на рис. 2.1 приведён график совместной плотности при $P(1) = 0,4$, $P(-1) = 0,6$, $\mu_1 = 1$, $\sigma_1 = 1$, $\mu_{-1} = 3$, $\sigma_{-1} = 2$.

Оптимальная (или байесовская) решающая функция будет выглядеть следующим образом

$$f^*(x) = \arg \max_y \varphi(x, y). \quad (2.1)$$

При $\varphi(x, 1) = \varphi(x, -1)$ функция $\arg \max$ возвращает оба значения, т.е. множество $\{-1, 1\}$. В этом случае значение байесовской решающей функции может выбираться произвольно, как 1 так и -1 , причём при практической реализации часто имеет смысл делать случайный (равновероятный) выбор между этими значениями.

Заметим, что в определении байесовской решающей функции 2.1 формула Байеса не использовалась, поэтому термин «байесовская», пожалуй, является не вполне удачным, однако, он устоявшийся, и мы будем его использовать.

На рис. 2.1 компоненты совместной плотности имеют две точки пересечения: при $x = a \approx -1,26$ и при $x = b \approx 1,93$. Оптимальная решающая функция

$$f^*(x) = \begin{cases} 1, & x \in [a, b] \\ -1, & x \notin [a, b] \end{cases}.$$

Вероятность ошибочной классификации для оптимального решения есть

$$R^* = P(f^*(x)) = \int_{f^*(x)=-1}^{\varphi(x, 1)} dx + \int_{f^*(x)=1}^{\varphi(x, -1)} dx.$$

Для приведённого примера

$$\begin{aligned} R^* &= \int_{-\infty}^a \varphi(x, 1) dx + \int_a^b \varphi(x, -1) dx + \int_b^{\infty} \varphi(x, 1) dx = \\ &\quad \frac{1}{2} P(1) \left(1 + \Phi \left(\frac{a - \mu_1}{\sigma_1} \right) + 1 - \Phi \left(\frac{b - \mu_1}{\sigma_1} \right) \right) + \\ &\quad \frac{1}{2} P(-1) \left(\Phi \left(\frac{b - \mu_{-1}}{\sigma_{-1}} \right) - \Phi \left(\frac{a - \mu_{-1}}{\sigma_{-1}} \right) \right) \approx 0,24. \end{aligned}$$

Полученное значение есть площадь заштрихованной области на рис. 2.1.

Здесь интегралы от нормальной плотности выражены через так называемую функцию Лапласа

$$\Phi(z) = \frac{2}{\sqrt{2\pi}} \int_0^z e^{-\frac{t^2}{2}} dt,$$

значения которой могут быть найдены в справочных таблицах либо вычислены на компьютере.

Введём функции условной вероятности классов

$$g_1(x) = \mathsf{P}(y = 1 | x) = \varphi_1(x) \frac{P_1}{\varphi(x)}, \quad g_{-1}(x) = 1 - g_1(x),$$

где

$$\varphi(x) = \varphi_1(x)P_1 + \varphi_{-1}(x)P_{-1}.$$

Здесь использована формула Байеса.

Байесовская решающая функция может быть записана как

$$f^*(x) = \arg \max_y g_y(x).$$

Если максимум достигается на обоих классах, можем выбрать любой, например, класс 1.

Поскольку функции $g_1(x)$ и $g_{-1}(x)$ в рассматриваемом случае положительны, их сравнение друг с другом эквивалентно сравнению их отношения с единицей, что, в свою очередь, эквивалентно сравнению логарифма отношения с нулюм, поэтому байесовскую решающую функцию можно представить в виде

$$f^*(x) = \begin{cases} 1, & l(x) \geq 0 \\ -1, & l(x) < 0 \end{cases},$$

где

$$l(x) = \ln \frac{g_1(x)}{g_{-1}(x)} = \ln \varphi_1(x) - \ln \varphi_{-1}(x) + \ln \frac{P(1)}{P(-1)}.$$

Функцию $l(x)$ будем называть разделяющей функцией.

Мы также можем использовать стандартную функцию знака (сигнум)

$$\text{sign}(x) = \begin{cases} 1, & x > 0 \\ 0, & x = 0 \\ -1, & x < 0 \end{cases}.$$

Можем записать $f^*(x) = \text{sign}(l(x))$. В этом случае мы допускаем, что на границе классов прогноз будет 0, что можно интерпретировать как «отказ от прогноза». Это ни на что не влияет, поскольку происходит на множестве меры 0.

Подставляя нормальные плотности, получаем

$$l(x) = \frac{(x - \mu_{-1})^2}{(\sigma_{-1})^2} - \frac{(x - \mu_1)^2}{(\sigma_1)^2} + \ln \frac{\sigma_{-1}}{\sigma_1} + \ln \frac{P(1)}{P(-1)}.$$

Видим, что $l(x) = 0$ представляет собой квадратное уравнение, которое имеет либо два корня, либо один корень — в случае $\sigma_1 = \sigma_{-1}$, либо не имеет корней — если одна из компонент совместной плотности целиком лежит под другой (также возможен очевидный вырожденный случай бесконечного множества корней — при полном совпадении компонент).

Заметим, что имеет место следующее соотношение

$$g_1(x) = P(y = 1|x) = \frac{1}{1 + e^{-l(x)}} = \sigma(l(x)).$$

Здесь $\sigma(z) = \frac{1}{1+e^{-z}}$ — так называемая логистическая функция (иногда также называемая сигмоидом или логит-функцией). Данная функция, как мы увидим в дальнейшем, очень часто возникает при решении задачи классификации и играет важную роль.

2.1.2 Многомерный случай

Рассмотрим теперь случай нескольких переменных X_1, \dots, X_n .

Условные плотности вероятности $\varphi_y(x)$ предполагаются нормальными и задаются параметрами μ_y и λ_y , где μ_y — вектор средних (математических ожиданий) для класса y , а λ_y — ковариационная матрица.

Оптимальная решающая функция записывается, так же как и в одномерном случае, через разделяющую функцию

$$l(x) = \ln \varphi_1(x) - \ln \varphi_{-1}(x) + \ln \frac{P(1)}{P(-1)}.$$

Решение уравнения $l(x) = 0$ задает границу между классами и называется разделяющей поверхностью.

Подставим в разделяющую функцию аналитические выражения для нормальных плотностей

$$\varphi_y(x) = \frac{1}{(2\pi)^{n/2} |\lambda_y|^{n/2}} e^{-\frac{1}{2} Q_y(x)},$$

где $Q_y(x) = (x - \mu_y)'(\lambda_y)^{-1}(x - \mu_y)$.

Штрих в записи квадратичной формы обозначает транспонирование вектора.

После преобразований имеем

$$2l(x) = Q_{-1}(x) - Q_1(x) + \ln |\lambda_{-1}| - \ln |\lambda_1| + 2 \ln P(1) - 2 \ln P(-1).$$

Далее преобразуем

$$\begin{aligned} Q_{-1}(x) - Q_1(x) &= \\ (x - \mu_{-1})'(\lambda_{-1})^{-1}(x - \mu_{-1}) - (x - \mu_1)'(\lambda_1)^{-1}(x - \mu_1) &= \\ x'((\lambda_{-1})^{-1} - (\lambda_1)^{-1})x - (\mu_{-1}'(\lambda_{-1})^{-1} - \mu_1'(\lambda_1)^{-1})x &- \\ x'((\lambda_{-1})^{-1}\mu_{-1} - (\lambda_1)^{-1}\mu_1) &+ \\ \mu_{-1}'(\lambda_{-1})^{-1}\mu_{-1} - \mu_1'(\lambda_1)^{-1}\mu_1. \end{aligned}$$

Наконец получаем

$$2l(x) = x'Ax + bx + c, \quad (2.2)$$

где

$$\begin{aligned} A &= (\lambda_{-1})^{-1} - (\lambda_1)^{-1}, \\ b &= 2\mu_1(\lambda_{-1})^{-1} - 2\mu_{-1}(\lambda_1)^{-1}, \\ c &= \mu_{-1}'(\lambda_{-1})^{-1}\mu_{-1} - \mu_1'(\lambda_1)^{-1}\mu_1 \\ &\quad + \ln \frac{|\lambda_{-1}|}{|\lambda_1|} + 2 \ln \frac{P_1}{P_{-1}}. \end{aligned}$$

При получении выражения для b использована коммутативность скалярного произведения векторов и коммутативность произведения вектора на симметричную матрицу (напомним, что ковариационная матрица симметрична).

Уравнение $l(x) = 0$ в общем случае задает поверхность второго порядка, которая при $\lambda_1 = \lambda_{-1}$, то есть при равенстве ковариационных матриц, вырождается в гиперплоскость.

Заметим, что ковариационные матрицы являются положительно определенными, ввиду чего линии (поверхности) уровня нормального распределения представляют собой эллипсоиды. При

этом разделяющая поверхность может оказаться любой поверхностью второго порядка: эллипсоидом, гиперболоидом, параболоидом, конусом, цилиндром.

2.1.3 Оценивание параметров

Пусть имеется выборка, представленная прямоугольной матрицей

$$S = ((x^i, y^i) | i = 1, \dots, N), \quad x^i = (x_1^i, \dots, x_n^i).$$

Параметр N представляет собой число независимых реализаций системы случайных величин и называется объемом выборки, n – число переменных.

Сформируем множество индексов (номеров) объектов, принадлежащих первому классу: $I_1 = \{i | y^i = 1\}$. Аналогично, $I_{-1} = \{i | y^i = -1\}$. Тогда $N_y = |I_y|$ есть число объектов класса y в выборке, $N_1 + N_{-1} = N$.

Проще всего оценить априорные (безусловные) вероятности классов, а именно: $\tilde{P}_y = \frac{N_y}{N}$, где \tilde{P}_y – оценка для P_y .

Обозначим через $\tilde{\mu}_y$ оценку вектора μ_y .

Компоненты вектора средних и ковариационной матрицы для каждого класса оцениваются по формулам

$$\begin{aligned} \tilde{\mu}_y &= \frac{1}{N_y} \sum_{i \in I_y} x^i, \\ \tilde{\lambda}_y &= \frac{1}{N_y} \sum_{i \in I_y} (x^i - \tilde{\mu}_y)(x^i - \tilde{\mu}_y)' . \end{aligned} \quad (2.3)$$

где I_y – множество индексов объектов класса y из выборки.

Здесь произведение векторов вычисляется по правилу умножение матриц, причём (поскольку транспонирован правый сомножитель) умножается вектор-столбец на вектор-строку, в результате чего получается квадратная матрица оценок ковариаций.

Оценки 2.3 являются оценками максимального правдоподобия. Можно использовать несмешённую версию оценок, но на типичных размерах выборки различие пренебрежимо.

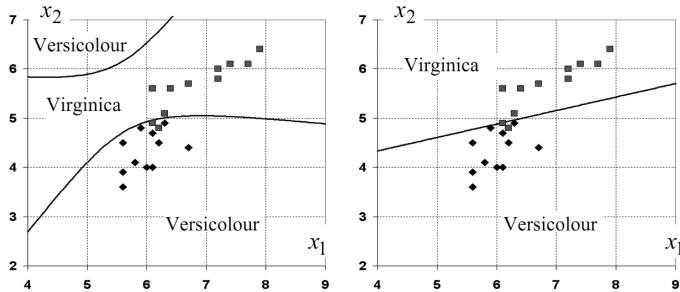


Рис. 2.2: Квадратичный и линейный дискриминант в задаче Iris.

Для построения выборочной разделяющей поверхности достаточно теперь в уравнение подставить вместо параметров распределений найденные оценки.

Случай равных матриц ковариации.

Если известно, что распределения обоих классов имеют одинаковые матрицы ковариации, то нужно всю выборку использовать для вычисления оценки $\tilde{\lambda}$ матрицы $\lambda = \lambda_1 = \lambda_{-1}$. Оценка находится следующим образом

$$\tilde{\lambda} = \frac{1}{N} \sum_{y \in \{1, -1\}} \sum_{i \in I_y} (x^i - \tilde{\mu}_y)(x^i - \tilde{\mu}_y)'. \quad (2.4)$$

Остальные параметры оцениваются так же, как в общем случае.

В качестве примера рассмотрим задачу об ирисах, изложенную в главе 2.

Поскольку мы разобрали метод применительно к случаю двух классов, выберем для классификации два вида ирисов: Iris Versicolour и Iris Virginica.

Если предположить ковариационные матрицы классов различными, то, оценив параметры распределения, получим разделяющую кривую, которая оказалась гиперболой, график которой приведён на левой диаграмме рис. 2.2.

Если предположить равенство ковариационных матриц, то получим разделяющую прямую, график которой приведён на правой диаграмме рис. 2.2.

На практике обычно нет информации о том, являются ли ковариационные матрицы классов одинаковыми, поэтому решение о способе их оценивания приходится принимать эмпирически.

Можно, например, применить статистический критерий проверки гипотезы о равенстве ковариационных матриц. Но на практике это нецелесообразно.

Раумный вариант: при большом объеме выборки оценивать матрицы для каждого класса отдельно, при малых выборках — оценивать одну матрицу и строить линейную разделяющую поверхность.

Дело в том, что квадратичный дискриминант требует оценить порядка n^2 параметров (напомним, что n — число переменных), а линейный дискриминант — порядка n параметров. Может показаться, что линейный дискриминант тоже оценивает ковариационную матрицу (где n^2 значений), но на самом деле важно именно число степеней свободы в решении, а линейное решение определяется уравнением разделяющей гиперплоскости, т.е. имеет $n + 1$ параметров.

Здесь мы впервые сталкиваемся с эффектом, что при малых выборках следует строить более простое решающее правило, даже если это упрощение вносит заведомую погрешность. В дальнейшем такая ситуация будет частой.

Решение о выборе между линейным и квадратичным дискриминантом целесообразно принимать на основе кросс-валидации.

Таким образом, даже казалось бы в простейшем случае, когда форма распределений известна, а именно нормальна, оптимальный метод построения решающей функции по конечной выборке неизвестен. При «малых» выборках мы строим линейный дискриминант (даже если точно знаем, что ковариационные матрицы не равны), при «больших» — квадратичный. Кажется, что в подобной ситуации должны быть и промежуточные варианты.

2.1.4 Фильтрация нетипичных значений

Описанный метод сильно чувствителен к «выбросам», т.е. нетипичным значениям переменных, сильно отличающимся от нормальных. Такие значения могут быть вызваны как ошибками из-

мерений (или ошибками переноса значений в таблицу), так и наличием нехарактерных объектов в выборке (например, растение повреждено вредителями и не выросло до обычных размеров).

Для решения проблемы выбросов существуют отработанные методы робастных оценок. Общая идея таких оценок заключается в следующем. Сначала строятся оценки параметров по всей выборке. После чего вся выборка упорядочивается по значению правдоподобия, рассчитанного по этим параметрам. Далее некоторая доля (например 1%) наименее правдоподобных точек выборки отбрасывается, после чего параметры вновь оцениваются по оставшимся объектам. Эту процедуру можно повторять несколько раз.

Такой алгоритм является достаточно универсальным (может применяться с разными методами) и позволяет эффективно отфильтровывать нетипичные значения в данных. Следует только иметь в виду, что после такой фильтрации в оценки условных вероятностей классов нужно делать поправки. Грубо говоря, если мы удалили из выборки 1% объектов, то ни в одной точке пространства мы не должны давать для функции $g_y(x)$ оценку меньше 0,01 и больше 0,99.

2.1.5 Свойства метода

Можно отметить следующие особенности рассмотренного метода:

- основан на предположении нормальности распределений,
- чувствителен к «выбросам», но эта проблема решаема,
- имеет низкую вычислительную трудоёмкость, решение в аналитическом виде,
- выбор между линейным и квадратичным решениями не однозначен.

Первую особенность как правило позиционируют как недостаток метода. Однако её можно считать, наоборот, достоинством. Аргументация этого следующая.

Если в какой-то задаче мы знаем, что распределения классов нормальны, это, очевидно, является некоторым обоснованием применения метода (линейного или квадратичного дискриминанта).

Если же мы не имеем оснований полагать распределения нормальными, или даже точно знаем, что они не являются нормальными, то из этого совершенно не следует, что данный метод в такой задаче работать не будет.

Это совершенно общее соображение: если что-то работает при некоторых предположениях, то это не значит, что оно не работает при нарушении предположений. Достаточность условий не подразумевает их необходимость.

Если распределения не являются нормальными, мы всё равно можем применять линейный и квадратичный дискриминант, но уже как чисто эвристический метод.

Заметим, что почти все (другие) методы машинного обучения изначально являются эвристическими. Это не значит, что они не имеют обоснования. Про обоснованность методов машинного обучения будем говорить в седьмой главе.

2.2 Дискриминант Фишера

2.2.1 Идея метода

Идея метода заключается в выборе направления, при проецировании выборки на которое образы классов оказываются наиболее удалёнными друг от друга.

Это выражается в максимизации критерия

$$\Phi(w) = \frac{(\tilde{\mu}_{w,1} - \tilde{\mu}_{w,-1})^2}{\tilde{S}_{w,1} + \tilde{S}_{w,-1}}, \quad (2.5)$$

где $\tilde{\mu}_{w,y} = \frac{1}{N_y} \sum_{i \in I_y} w x_i$ – среднее,

а $\tilde{S}_{w,y} = \frac{1}{N_y} \sum_{i \in I_y} (w x_i - \tilde{\mu}_{w,y})^2$ – средний квадрат отклонений проекций.

Этот критерий имеет очень простой содержательный смысл: два множества точек тем дальше друг от друга, чем дальше их

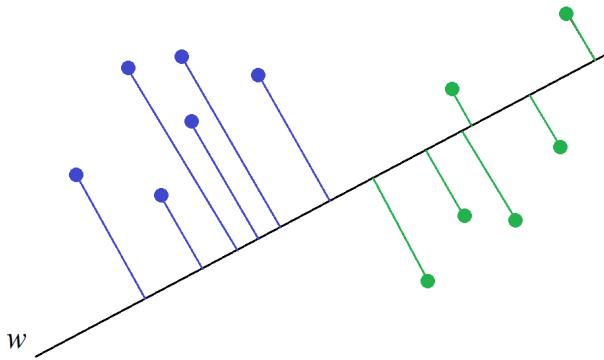


Рис. 2.3: Пример проецирования выборки на направление.

средние и чем меньше разброс точек.

Требуется найти оптимальное направление.

Оказывается, что в 2.5 максимум $\Phi(w)$ достигается при

$$w_\Phi = \tilde{S}^{-1}(\tilde{\mu}_1 - \tilde{\mu}_{-1}), \quad (2.6)$$

где $\tilde{\mu}_y$ – среднее точек выборки y -го класса, а \tilde{S}_y – выборочная ковариационная матрица y -го класса, $\tilde{S} = \tilde{S}_1 + \tilde{S}_{-1}$.

Полученное выражение неожиданно оказывается практически совпадающим с 2.2 при равных матрицах ковариаций.

Отличия незначительны, но их стоит отметить.

Дискриминант Фишера не предполагает равенства ковариационных матриц и использует усреднённую матрицу по классам. В отличие от 2.4, классы здесь дают вклад с равными весами, независимо от соотношения классов в выборке.

На самом деле это свойство можно изменить, скорректировав критерий 2.5, но это бы противоречило идеи критерия. Здесь мы не предполагаем равенство ковариационных матриц, поэтому оцениваем каждую отдельно, а потом усредняем (с равным весом). А в линейном дискриминанте предполагаем равенство матриц, поэтому оцениваем одну матрицу по полной выборке.

Другое отличие в том, что дискриминант Фишера даёт только направление, но не описывает способ постановки границы между проекциями классов. Предполагается, что если задача сведена к одномерной, то разумным образом разделить классы не составит труда (например, непосредственной минимизацией числа ошибок).

2.2.2 Свойства метода

Замечания.

- Метод не требует никаких вероятностных предположений.
- Выражение для w_F очень похоже на выражение для нормали к разделяющей гиперплоскости для случая нормальных распределений с равными матрицами ковариаций.
- Метод предполагает оценивание по выборке только $n + 1$ параметров и не требователен к объёму выборки.
- Метод неустойчив к «выбросам», но позволяет использовать приём робастного оценивания.

Мы получили нетривиальный факт, что метод, выведенный из очень сильных предположений (нормальность и равенство ковариаций) можно получить вообще без всяких предположений, только лишь на основе геометрических эвристик.

Это позволяет сделать вывод, что метод, выведенный при сильных предположениях, может оставаться пригодным и при нарушении этих предположений.

2.3 Логистическая регрессия

2.3.1 Идея метода

Рассмотрим функцию условной вероятности для класса 1

$$g(x) = P(y = 1|x) = \frac{P(1)\varphi_1(x)}{P(1)\varphi_1(x) + P(-1)\varphi_{-1}(x)} = \frac{1}{1 + e^{-l(x)}},$$

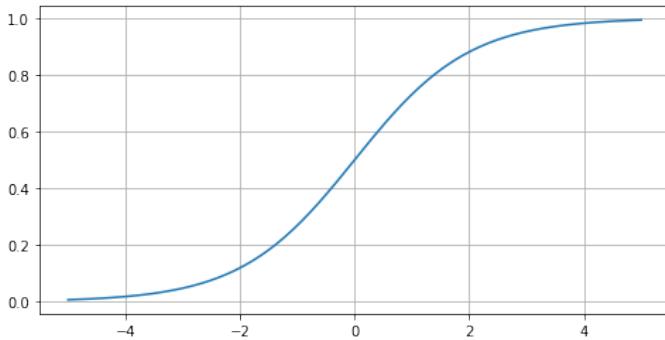


Рис. 2.4: График функции $\sigma(z)$.

где $l(x)$ – разделяющая функция.

Подставив нормальные плотности при равных матрицах ковариаций, получим

$$g(x) = \frac{1}{1+e^{-(wx+w_0)}} = \sigma(wx+w_0),$$

где w и w_0 есть переобозначение параметров b и c соответственно из линейного дискриминанта 2.2.

Здесь $\frac{1}{1+e^{-z}}$ – так называемая логистическая функция (иногда также называемая сигмоидом или логит-функцией).

Определить (полную) вероятностную модель данных мы можем либо задав совместную плотность $\varphi(x, y)$, либо задав пару $\varphi_y(x)$, $P(y)$, либо задав пару $P(y|x)$, $\varphi(x)$. Все способы эквивалентны. Два последних варианта переводятся друг в друга формулой Байеса.

В методе квадратичного дискриминанта мы предполагали нормальность $\varphi_y(x)$ и затем находили $g_y(x) = P(y|x)$.

Возникает разумный вопрос: зачем делать предположения о форме $\varphi_y(x)$, если нам нужна $g_y(x)$. Почему не сделать сразу предположение о форме $g_y(x)$?

Оказывается, что так сделать можно и именно в этом заключается идея логистической регрессии.

При этом форма безусловной плотности $\varphi(x)$ нас не интересует, и мы о ней делать предположений не будем. В этом смысле

мы будем строить неполную вероятностную модель. Эта модель ничего не говорит о вероятности появления (правдоподобии) x , но описывает условную вероятность $P(y|x)$.

Параметры такой модели оказывается возможным оценивать уже привычным методом максимального правдоподобия. Мы подбираем такие параметры, чтобы вероятность имеющейся выборки была максимальна.

2.3.2 Оценивание параметров

Метод логистической регрессии основан на оценивании функции условной вероятности моделью $\tilde{g}(x) = \sigma(\tilde{w}x + \tilde{w}_0)$, в которой \tilde{w} и \tilde{w}_0 – настраиваемые параметры.

На практике параметры модели обычно оцениваются путём максимизации критерия правдоподобия (условной вероятности выборки).

$$-K_S(\tilde{w}, \tilde{w}_0) = \sum_{i \in I_1} \ln \tilde{g}(x^i) + \sum_{i \in I_{-1}} \ln(1 - \tilde{g}(x^i)).$$

В случае линейно разделимой выборки максимум правдоподобия достигается на ступенчатой функции, которая является пределом сигмоида, когда $w^2 \rightarrow \infty$.

Формально в классе сигмоидов максимум не достигается, поэтому алгоритм численной оптимизации не сойдётся.

Чтобы обеспечить как существование решения, так и его единственность, требуется регуляризация. Это некоторая добавка к критерию, обеспечивающая устойчивость решения.

В данной задаче такой добавкой может быть $\varkappa(w^2)$ – некоторая монотонная функция от нормы вектора направления.

2.3.3 Функция потерь

Мы впервые встретились с методом, который сводится к минимизации аддитивной функции выборки. Дальнейшие методы именно так и будут устроены.

Метод минимизирует функцию

$$K_S(\tilde{w}, \tilde{w}_0) = \varkappa(w^2) + \sum_{i=1}^N L(\sigma(\tilde{w}x^i + \tilde{w}_0), y^i), \quad (2.7)$$

где

$$L(p, y) = \begin{cases} -\ln(p), & y=1 \\ -\ln(1-p), & y=-1 \end{cases} \quad (2.8)$$

представляет собой логарифмическую функцию потерь.

Учитывая, что $\ln \sigma(-z) = \ln(1 - \sigma(z))$, выражение 2.7 можно преобразовать к виду

$$K_S(\tilde{w}, \tilde{w}_0) = \varkappa(w^2) + \sum_{i=1}^N -\ln(\sigma(y^i \cdot (\tilde{w}x^i + \tilde{w}_0))). \quad (2.9)$$

Вид получившейся функции потерь приведён на рис. 2.7

2.3.4 Свойства метода

Замечания.

- Метод логистической регрессии похож на линейный дискриминант, но ослабляет вероятностные предположения.
- На практике (почти) никто вероятностные предположения не проверяет.
- По сравнению с линейным дискриминантом метод логистической регрессии более устойчив в «выбросам».
- Для повышения устойчивости требуется регуляризатор.

Можно было ожидать, что логистическая регрессия окажется похожа на линейный дискриминант. Но она окажется больше похожа на метод опорных векторов.

2.4 Наивный байесовский классификатор

«Наивный» байесовский классификатор основан на предположении условной независимости переменных. Данное предположение редко обосновано в реальных задачах, поэтому метод и получил такое название.

2.4.1 Случай двух классов

Из формулы Байеса можем записать

$$g(x) = P(y = 1|x) = \frac{P(dx, y = 1)}{P(dx, y = 1) + P(dx, y = -1)} = \frac{1}{1 + \frac{1-p}{p} \cdot \frac{P(dx|y=-1)}{P(dx|y=1)}},$$

где $p = P(y = 1)$.

Данное выражение можно преобразовать в виду

$$g(x) = \sigma \left(z_0 + \ln \frac{P(dx|y = 1)}{P(dx|y = -1)} \right),$$

где введено обозначение $z_0 = \ln p - \ln(1 - p) = \sigma^{-1}(p)$, а $\sigma(z) = \frac{1}{1+e^{-z}}$ — уже знакомая нам логистическая функция (сигмоид).

Выразим правую часть

$$\ln \frac{P(dx|y = 1)}{P(dx|y = -1)} = \sigma^{-1}(g(x)) - z_0. \quad (2.10)$$

Для ряда читателей, возможно, окажется непривычной используемая здесь нотация в духе производных Радона-Никодима (производная по мере). В этом случае можно при чтении везде мысленно заменить $P(dx...)$ на плотность вероятности $\varphi(x...)$. Если непонятно, как это сделать, можно посмотреть выкладки следующего подраздела, где как раз используются плотности. Там мы выведем формулу для нескольких классов, из которых в частном случае получается то, что выведем сейчас.

Смысл использования производных по мере исключительно в том, чтобы сделать выкладки универсальными, а именно пригодными для случаев, когда пространство переменных образуют случайные величины любых типов: дискретные, (абсолютно)

непрерывные, сингулярные. В противном случае для дискретных и непрерывных величин потребуются отдельные выражения.

Поскольку переменные разных типов — типичная ситуация в анализе данных, оказывается полезным изучить такие конструкции как производные по мере и интегралы Лебега и Лебега-Стилтьеса.

Независимость переменных означает, что совместное распределение раскладывается в произведение маргинальных распределений (т.е. одномерных распределений по каждой переменной).

Поэтому для (условно) независимых переменных имеем

$$\frac{P(dx|y=1)}{P(dx|y=-1)} = \prod_{j=1}^n \frac{P(dx_j|y=1)}{P(dx_j|y=-1)}.$$

После логарифмирования получим равносильное выражение

$$\ln \frac{P(dx|y=1)}{P(dx|y=-1)} = \sum_{j=1}^n \ln \frac{P(dx_j|y=1)}{P(dx_j|y=-1)}.$$

Выразим обе части этого равенства через $g(x)$ и $g_j(x)$.

Аналогично формуле 2.10 мы можем вывести выражение для функций $g_j(x_j) = P(y=1|x_j)$, а именно

$$\ln \frac{P(dx_j|y=-1)}{P(dx_j|y=1)} = \sigma^{-1}(g_j(x)) - z_0.$$

Собрав всё вместе, получаем

$$\sigma^{-1}(g(x)) - z_0 = \sum_{j=1}^n (\sigma^{-1}(g_j(x_j)) - z_0). \quad (2.11)$$

Если ввести новые переменные $z_j = \sigma^{-1}(g_j(x_j)) - z_0$, то в пространстве этих переменных решающая функция будет иметь вид логистической регрессии

$$g(x) = \sigma \left(w_0 + \sum_{j=1}^n w_j z_j \right), \quad (2.12)$$

при $w_0 = z_0$, $w_j \equiv 1$.

Предположение о независимости переменных на самом деле редко выполняется, поэтому метод, на нём основанный, действительно может считаться наивным. Однако различные варианты частичной независимости переменных вполне типичны для реальных задач. Мы можем «ослабить» гипотезу независимости, полагая w_j в 2.12 свободными параметрами (и подбирая их самым обычным методом логистической регрессии).

Такая замена переменных (x_j на z_j) называется target encoding (хорошего перевода нет).

Приём target encoding, т.е. кодирование значений категориальной переменной преобразованной прогнозируемой вероятностью, является одной из стандартных техник машинного обучения. Мы можем на переменных z_j использовать не только логистическую регрессию, но и вообще любой метод машинного обучения.

2.4.2 Случай нескольких классов

В большинстве случаев мы не рассматриваем специально случай нескольких классов, поскольку для большинства методов переход от двух классов ко многим осуществляется типовыми приёмами, которые мы рассмотрим в главе, специально посвящённой общим приёмам.

Однако для рассматриваемого метода рассмотрение нескольких классов приводит к важным формулам.

Итак, цель — выразить условную вероятность заданного класса при условии конкретных значений всех переменных через условные вероятности этого класса по каждой переменной, т.е. требуется выразить $P(y = \omega|x)$ через $P(y = \omega|x_j)$, $j = 1, \dots, n$.

Введём обозначения

$$u_j^\omega = \ln P(y = \omega|x_j) - \ln P(y = \omega).$$

Для простоты выкладок предположим, что все переменные X_j — абсолютно непрерывные случайные величины, т.е., что существует (многомерная) плотность вероятности $\varphi(x)$. Для произвольных случайных величин выкладки требуют привлечения производных по мере и читатель может провести их самостоятельно по аналогии с выкладками предыдущего раздела.

Из формулы Байеса можем записать

$$\mathbb{P}(y = \omega|x) = \frac{\varphi_\omega(x)P(\omega)}{\varphi(x)},$$

где $\varphi_\omega(x)$ – условная плотность вероятности, $P(\omega)$ – вероятность класса ω .

По предположению о независимости переменных

$$\varphi_\omega(x) = \prod_{j=1}^n \varphi_{\omega,j}(x_j).$$

Ещё раз применим формулу Байеса:

$$\varphi_{\omega,j}(x_j) = \frac{\mathbb{P}(y = \omega|x_j)\varphi_j(x_j)}{P(\omega)}.$$

Соберём всё воедино

$$\mathbb{P}(y = \omega|x) = \frac{P(\omega)}{\varphi(x)} \prod_{j=1}^n \frac{\mathbb{P}(y = \omega|x_j)\varphi_j(x_j)}{P(\omega)}.$$

Обозначим

$$C(x) = \frac{1}{\varphi(x)} \prod_{j=1}^n \varphi_j(x_j)$$

и

$$\begin{aligned} \zeta_\omega(x) &= \ln P(\omega) + \sum_{j=1}^n (\ln \mathbb{P}(y = \omega|x_j) - \ln P(\omega)) = \\ &\quad \ln P(\omega) + \sum_{j=1}^n u_j^\omega. \end{aligned}$$

В этих обозначениях

$$\mathbb{P}(y = \omega|x) = C(x)e^{\zeta_\omega(x)}.$$

Величина $C(x)$ нам неизвестна. К счастью, её можно найти из условия, что сумма вероятностей всех классов (при любом условии) равна 1. Получаем

$$P(y = \omega|x) = \frac{e^{\zeta_\omega(x)}}{\sum_{v=1}^k e^{\zeta_v(x)}}. \quad (2.13)$$

Цель достигнута.

Выражение, что мы получили, есть ни что иное как хорошо известная (например из нейросетей) функция softmax, которая является обобщением логистической функции.

Убедимся, что в случае двух классов выражение 2.13 превращается в 2.11. Имеем

$$P(y = 1|x) = \frac{e^{\zeta_1(x)}}{e^{\zeta_1(x)} + e^{\zeta_{-1}(x)}} = \frac{1}{1 + e^{\zeta_{-1}(x) - \zeta_1(x)}} = \sigma(\zeta_1(x) - \zeta_{-1}(x)).$$

При этом

$$\zeta_1(x) - \zeta_{-1}(x) = \ln P(1) - \ln P(-1) + \sum_{j=1}^n (u_j^1 - u_j^{-1}).$$

Заметив, что $u_j^1 - u_j^{-1} = \sigma^{-1}(g_j(x_j))$, получаем искомое.

2.4.3 Ряд Бахадура

Рассмотрим интересный подход, касающийся случая бинарных (двоичных) переменных.

Бинарными называются переменные, принимающие одно из двух значений: «истина» и «ложь», которые обозначаются соответственно 1 и 0.

Рассмотрим набор переменных X_1, \dots, X_n со значениями $x = (x_1, \dots, x_n)$, $x_j \in \{0, 1\}$, $x \in X = \{0, 1\}^n$.

Если считать переменные независимыми, то для задания вероятностной модели требуется n параметров, а в общем случае модель совместного распределения имеет $2^n - 1$ параметров. Если объём выборки существенно больше 2^n , то оправдано оценивать совместное распределение, если объём выборки сравним с n , то

имеет смысл использовать модель независимых переменных, даже если нет оснований ожидать, что переменные действительно независимы.

В данном разделе рассмотрим модель, которую уместно использовать, когда объём выборки много больше n , но меньше 2^n . Данная модель связана с разложением совместного распределения в ряд Бахадура.

Идея разложения в ряд Бахадура состоит в том, чтобы, начиная с приближения независимых переменных, последовательно учитывать парные зависимости переменных, зависимости в тройках, четвёрках и т.д. Разложение в ряд Бахадура выглядит следующим образом

$$P(x) = P^1(x) \sum_{j=1}^n q_j(x).$$

Здесь $P^1(x)$ – значение совместной вероятности в предположении независимости переменных, т.е.

$$P^1(x) = \prod_{j=1}^n P(x_j).$$

Единица сверху — верхний индекс.

Остальные компоненты определяются как

$$\begin{aligned} q_1(x) &\equiv 1, & q_2(x) &= \sum_{j=1}^n \sum_{k=j+1}^n \rho_{jk} z_j z_k, \\ q_3(x) &= \sum_{j=1}^n \sum_{k=j+1}^n \sum_{l=k+1}^n \rho_{jkl} z_j z_k z_l, & \dots . \end{aligned}$$

В этих выражениях использованы обозначения

$$z_j = \frac{x_j - p_j}{\sqrt{p_j(1-p_j)}},$$

где $p_j = P(x_j = 1)$, а

$$\rho_{jk} = \sum_{x \in X} z_j z_k P(x), \quad \rho_{jkl} = \sum_{x \in X} z_j z_k z_l P(x), \quad \dots .$$

Распределения

$$P^M(x) = P^1(x) \sum_{j=1}^M q_j(x).$$

представляют собой частичные суммы ряда Бахадура, при этом $P^2(x)$ учитывает только парные зависимости переменных, $P^3(x)$ учитывает парные зависимости и зависимости в тройках. Полная сумма ряда совпадает с исходным распределением $P(x)$.

Частичные суммы ряда Бахадура могут использоваться в качестве приближений для $P(x)$.

Чем больше членов ряда используется, тем больше параметров требуется оценить по выборке и тем выше сложность модели.

Следует выбирать сложность модели адекватной объёму выборки. Грубой рекомендацией здесь будет выбирать такую модель, чтобы число оцениваемых параметров было хотя бы в несколько раз меньше объёма выборки. Более точно выбрать сложность модели можно с помощью метода статистического моделирования.

Разложение в ряд Бахадура вполне может использоваться для решения задачи классификации в пространстве бинарных переменных.

Вместе с тем, подобные задачи встречаются довольно редко, поэтому описанное разложение относительно мало распространено на практике.

Тем не менее, знакомство с ним очень полезно для понимания механизмов, за счёт которых одни методы построения решающих функций оказываются эффективнее других. Мы ещё вернёмся к этому разложению при выяснении причин эффективности бустинга.

2.4.4 Свойства метода

Характеризуя «наивный» байесовский классификатор, можно сделать следующие замечания.

- Предположение независимости переменных эквивалентно аддитивности в пространстве преобразованных переменных.

- Полученное преобразование является разумным вариантом target encoding, может использоваться и для вещественных переменных.
- Наивный байесовский классификатор является частным случаем логистической регрессии с использованием target encoding.
- Предположение о (строгой) независимости переменных обычно оказывается неоправданно сильным, однако в ослабленном виде гипотеза независимости очень полезна.

Поскольку в реальных задачах предполагать независимость переменных как правило нет оснований, использовать «наивный» байесовский классификатор редко бывает целесообразным. Однако попробовать применить «ослабленный» вариант метода в виде логистической регрессии с target encoding — почти всегда оправдано. Если при этом коэффициенты регрессии окажутся близки к единице, то это может служить свидетельством независимости переменных (и тогда можно применить собственно наивный байесовский классификатор).

Отметим важный факт. Почти для всех методов машинного обучения имеет место правило, что чем больше переменных, тем больший объём выборки требуется для обучения (из-за чего даже появился термин «проклятие размерности»).

Исключением из этого правила является случай условно независимых переменных. Здесь всё наоборот: чем больше переменных, тем большую точность прогноза можно получить при том же объёме выборки. Это похоже на то, как привлечение множества независимых мнений позволяет сделать более точные выводы.

2.5 Метод опорных векторов

Метод был разработан ещё в 60-е годы (В.Н. Вапник и др.) и изначально назывался методом обобщённого портрета.

В 90-е годы метод был усовершенствован и стал называться машиной опорных векторов (support vector machine, SVM).

Как и дискриминант Фишера, метод опорных вектором основан на поиске такого направления в пространстве переменных, при проецировании на которое объекты разных классов бы оказались в наибольшей степени отделены друг от друга. Но если критерий Фишера отделение классов понимает как нормированное расстояние между средними, то SVM в качестве расстояния между множествами использует расстояние между ближайшими (крайними) точками. Направление выбирается так, чтобы максимально отдалить объекты от разделяющей поверхности.

2.5.1 Постановка задачи классификации

Введём необходимые обозначения.

Пусть $X = \mathbb{R}^n$ – пространство значений прогнозирующих переменных, $Y = \{-1, 1\}$ – значения прогнозируемой переменной, $D = X \times Y$.

Решающая функция (алгоритм классификации) есть отображение

$$f : X \rightarrow Y.$$

Пусть $S = \{(x^i, y^i) \in D \mid i = \overline{1, N}\}$ – случайная независимая выборка, $S \in D^N$.

Метод построения решающих функций $Q: D^N \rightarrow \Phi$ есть отображение множества выборок во множество решений (Φ – заданный класс решающих функций).

Чтобы задача имела смысл, нужно определить качество решения. Для этого вводится понятие целевой функции потерь $L(\cdot, \cdot)$, которая характеризует степень несоответствия прогноза и истинного класса.

В данном разделе будем понимать потери как несовпадение прогнозируемого и истинного класса, т.е. $L(y, f(x)) = I(y \neq f(x))$, где $I(\cdot)$ – индикаторная функция (возвращает 1, если выражение истинно, иначе – 0).

Эмпирическим риском называются средние потери на обучающей выборке

$$\tilde{R}(S, f) = \frac{1}{N} \sum_{i=1}^N L(y^i, f(x^i)).$$

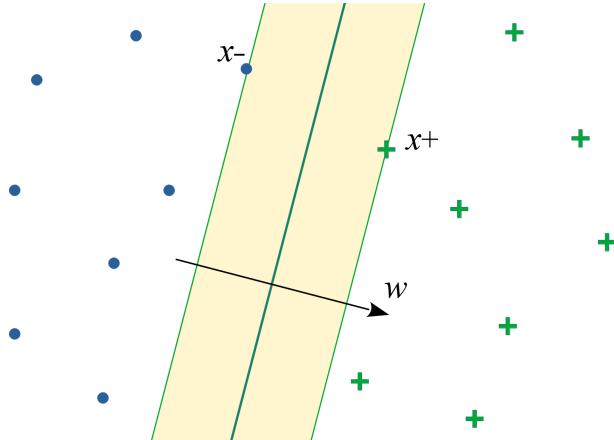


Рис. 2.5: Иллюстрация идеи метода SVM.

Решающую функцию будем искать в форме линейного порогового классификатора

$$f(x) = \text{sign}(wx - w_0).$$

Требуется найти вектор w и скаляр w_0 , минимизирующие эмпирический риск при дополнительных требованиях.

2.5.2 Линейно разделимая выборка

Предположим, что выборка оказалось такова, что существует гиперплоскость, которая разделяет классы. Это эквивалентно тому, что существует линейный классификатор с нулевым значением эмпирического риска.

Если существует гиперплоскость, безошибочно разделяющая выборку, то почти всегда её можно немного «подвинуть», так что выборка всё ещё разделится безошибочно. Получается бесконечное (континуальное) множество решений с нулевым эмпирическим риском. Нужно выбрать из этого множества «наилучший» вариант.

Критерием для такого выбора будет ширина разделяющей полосы. Если мы спроектировали точки на нормаль к гиперплоско-

сти, то ширина разделяющей полосы будет равна наименьшему расстоянию между проекциями точек разных классов. Пример построения разделяющей полосы для некоторого направления w показан на рис. 2.5.

Заметим, что для задания направления можно использовать вектор любой длины. Таким образом, длина вектора w — это «лишний» свободный параметр, от которого нужно избавиться, чтобы решение было единственным.

Наиболее очевидное решение — это рассматривать векторы единичной длины, но такой вариант не позволит эффективно решать задачу оптимизации. Удобнее оказывается использовать следующее условие нормировки

$$\min_{(x^i, y^i) \in V} y^i(x^i w - w_0) = 1. \quad (2.14)$$

Легко убедиться, что для любого направления w' , вдоль которого выборка классифицируется безошибочно, можно выбрать такие числа (скаляры) a и w_0 , что при $w = aw'$ условие 2.14 будет выполнено.

Минимум в 2.14 достигается на каких-то выборочных точках, которые назовём граничными. При этом ширина разделяющей полосы оказывается максимальной, если среди граничных точек оказываются представители обоих классов. Выберем два таких представителя: x_+ и x_- .

Разделяющая полоса в некотором смысле «опирается» на эти граничные точки, которые поэтому и называются опорными векторами.

При дополнительном требовании максимизации ширины разделяющей полосы условие 2.14 определяет однозначный выбор нормы направляющего вектора и константы w_0 .

Для граничных точек условие 2.14 превращается в

$$x_+ w - w_0 = 1, \quad -(x_- w - w_0) = 1.$$

Ширину разделяющей полосы легко выразить через координаты граничных точек

$$(x_+ - x_-) \cdot \frac{w}{|w|} = \frac{(w_0 + 1) - (w_0 - 1)}{|w|} = \frac{2}{|w|}.$$

Получается, что максимизация ширины разделяющей полосы при условии 2.14 эквивалентна минимизации нормы вектора w , или, что то же самое, минимизации его квадрата.

В результате поиск вектора w и скаляра w_0 сводится к решению следующей оптимизационной задачи

$$\begin{cases} w^2 \rightarrow \min_{w, w_0} \\ y^i(x^i w - w_0) \geq 1, \quad i = 1, \dots, N. \end{cases} \quad (2.15)$$

Границные условия в этой задаче эквивалентны условию 2.14 при дополнительном требовании минимизации нормы w .

Данная задача относится к хорошо известному классу задач квадратичной оптимизации. Для решения таких задач существуют готовые алгоритмы. Оптимизационную задачу можно решать методом активных ограничений (incremental active set method, INCAS), частным случаем которого является симплекс-метод.

2.5.3 Линейно неразделимая выборка

Очевидно, что линейная разделимость является очень обременительным ограничением, которое необходимо снять, чтобы метод имел приемлемую область применения.

Если выборка не является линейно разделимой, то не существует вектора w , который бы удовлетворял всем ограничениям задачи 2.15, т.е. ограничения получаются несовместными и задача не имеет решения.

Получается, что ограничения нужно ослабить. Для этого введём величины ξ_i , которые будут задавать «допустимую степень нарушения» каждого ограничения. Получим следующую задачу квадратичного программирования

$$\begin{cases} \frac{w^2}{2} + C \sum_{i=1}^N \xi_i \rightarrow \min_{w, w_0, \xi} \\ y^i(x^i w - w_0) \geq 1 - \xi_i, \quad i = 1, \dots, N, \\ \xi_i \geq 0, \quad i = 1, \dots, N, \end{cases} \quad (2.16)$$

где $C > 0$ – параметр. «Степень нарушения» не может быть отрицательной, поэтому накладываем условия $\xi_i \geq 0$.

Очевидно, что чем меньше ξ_i , тем лучше решение, поэтому сумму всех ξ_i мы добавили в целевую функцию, которая минимизируется. Чтобы управлять вкладом каждого слагаемого, введён параметр C .

Заметим, что для линейно разделимой выборки решение уже не обязательно даст нулевой эмпирический риск. При не очень больших C может оказаться выгоднее допустить нарушение разделяющей полосы или даже ошибочную классификацию некоторых объектов, но за счёт этого построить полосу большей ширины.

Задача 2.16 приближается к задаче для линейно разделимой выборки, когда $C \rightarrow \infty$.

Константу C обычно выбирают по критерию скользящего контроля.

2.5.4 Двойственная задача

Метод опорных векторов является одним из самых красивых в математическом плане методов машинного обучения. Наиболее изящные его возможности получаются с помощью перехода к двойственной задаче.

Чтобы разобраться в этой технике, попытаемся кратко и наглядно изложить суть некоторых идей математического программирования.

Рассмотрим простейшую постановку задачи на нахождение условного экстремума

$$\begin{cases} f(x) \rightarrow \min_x \\ \psi(x) = 0. \end{cases}$$

Эту задачу можно проиллюстрировать наглядно. Пусть имеется некоторый рельеф местности и некоторая траектория движения (тропинка). Требуется найти на этой тропинке наивысшую или самую низкую точку 2.6. Очевидно, что в таких точках тропинка будет идти параллельно склону, т.е. параллельность склону (или касание тропинкой некоторой линии уровня) — это необходимое условие экстремума.

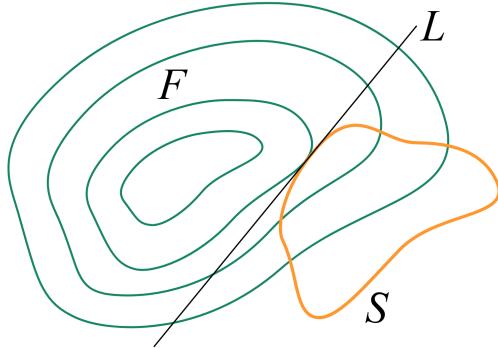


Рис. 2.6: Иллюстрация идеи множителей Лагранжа.

Математически параллельность склону означает параллельность векторов градиента высоты и нормали к траектории, т.е.

$$\frac{\partial f(x)}{\partial x} = \lambda \frac{\partial \psi(x)}{\partial x}.$$

Здесь λ имеет смысл коэффициента пропорциональности двух векторов.

После переноса в левую часть и выноса производной получаем в скобках функцию Лагранжа.

$$\frac{\partial(f(x) - \lambda\psi(x))}{\partial x} = \frac{\partial \mathfrak{L}(x)}{\partial x}.$$

Линии уровня и ограничение

Пусть теперь в задача на нахождение экстремума ограничение задано не равенством, а неравенством

$$\begin{cases} f(x) \rightarrow \min_x \\ \psi(x) \geq 0. \end{cases}$$

Тогда решение (искомый минимум) находится либо внутри области, либо на её границе. Второй случай мы только что рассмотрели, а внутри области необходимым условием экстремума является просто равенство градиента нулю.

Эти два случая можно свести в одну систему, которые называются условиями Каруша–Куна–Таккера. Необходимые условия экстремума

$$\begin{cases} \frac{\partial f(x)}{\partial x} = \lambda \frac{\partial \psi(x)}{\partial x}, \\ \lambda \psi(x) = 0, \\ \lambda \geq 0. \end{cases}$$

Второе условие называют условием дополняющей нежёсткости. Его смысл очень простой. Для его выполнения один из сомножителей должен быть нулевым. Если $\lambda = 0$, то первое уравнение (условие стационарности) превращается в равенство градиента нулю, и мы ищем безусловный экстремум. Если $\lambda \neq 0$, получаем условный экстремум.

Условие неотрицательности для λ означает, что если минимум находится на границе области, то градиент в этой точке должен быть направлен внутрь области (т.е. функция должна уменьшаться при движении «наружу» — иначе минимум окажется внутри).

Применим данный аппарат к задаче 2.16. Функция Лагранжа для SVM есть

$$\mathfrak{L}(w, w_0, \xi; \lambda, \eta) = \frac{w^2}{2} + C \sum_{i=1}^N \xi_i - \sum_{i=1}^N \lambda_i (y^i (x^i w - w_0) - 1 + \xi_i) - \sum_{i=1}^N \xi_i \eta_i.$$

После преобразований

$$\mathfrak{L}(w, w_0, \xi; \lambda, \eta) = \frac{w^2}{2} - \sum_{i=1}^N \lambda_i (y^i (x^i w - w_0) - 1) - \sum_{i=1}^N \xi_i (\lambda_i + \eta_i - C).$$

Дифференцируем (находим условия стационарности)

$$\begin{cases} \frac{\partial \mathfrak{L}}{\partial w} = 0, & w = \sum_{i=1}^N \lambda_i y^i x^i, \\ \frac{\partial \mathfrak{L}}{\partial w_0} = 0, & \sum_{i=1}^N \lambda_i y^i = 0, \\ \frac{\partial \mathfrak{L}}{\partial \xi_i} = 0, & \eta_i + \lambda_i = C, \quad i = 1, \dots, N. \end{cases}$$

Выразим всё через λ_i

$$-\mathcal{L}(\lambda) = \sum_{i=1}^N \lambda_i + \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y^i y^j x^i x^j$$

В результате получаем двойственную задачу

$$\begin{cases} \sum_{i=1}^N \lambda_i + \frac{1}{2} \sum_{i=1}^N \sum_{j=1}^N \lambda_i \lambda_j y^i y^j x^i x^j \rightarrow \min_{\lambda} \\ 0 \leq \lambda_i \leq C, \quad i = 1, \dots, N, \\ \sum_{i=1}^N \lambda_i y^i = 0. \end{cases} \quad (2.17)$$

Приведённые выкладки являются слишком краткими и поверхностными, чтобы по ним полноценно разобраться в выводе, поэтому рекомендуется углубленно изучить тему в специализированных источниках.

Для понимания дальнейших рассуждений достаточно просто принять как факт, что задача 2.17 равносильна (эквивалентна) задаче 2.16, поэтому для реализации метода SVM можно решать любую из них.

Точки, для которых $\lambda_i > 0$, называются опорными векторами.

2.5.5 Ядра и спрямляющие пространства

Задача 2.17 примечательна тем, что вектора x^i входят в выражения только через скалярные произведения друг с другом. Поэтому если мы знаем скалярные произведения, то непосредственно компоненты векторов нам уже не нужны.

Поскольку $w = \sum_{i=1}^N \lambda_i y^i x^i$, итоговый классификатор имеет вид

$$f(x) = \text{sign} \left(\sum_{i=1}^N \lambda_i y^i x^i - w_0 \right). \quad (2.18)$$

Здесь тоже используются только скалярные произведения, но не компоненты векторов.

Получается, что если мы знаем скалярные произведения между векторами, описывающими объекты, то сама таблица данных не нужна.

Это позволяет решать так называемую задачу беспризнаковой классификации. Её мы обсудим чуть позже, поскольку сначала нужно ввести понятие функции ядра.

Идея ядра возникла из желания снять ограничение линейности решения.

Есть много задач, где решающая функция должна быть нелинейной. При этом функция может быть простой. Например, объекты первого класса находятся все в некотором круге, а другой класс — снаружи круга. Линейной функцией в исходном пространстве такие данные не разделить.

Однако, есть такой типовой приём как feature engineering, который заключается в конструировании новых переменных как функций от исходных. Если мы добавим к исходным переменным их квадраты, то линейная функция в новом пространстве будет квадратичной функцией исходных переменных (в частности, может отделить круг).

Добавление новых переменных есть переход от исходного пространства X к новому пространству H с помощью некоторого преобразования $\psi : X \rightarrow H$. Пространство H называют спрямляющим.

Теперь отметим простой, но важный факт: скалярное произведение в спрямляющем пространстве есть некоторая функция от исходных переменных

$$K(x, x') = \psi(x) \cdot \psi(x').$$

Такие функции будем называть ядрами.

Функция ядра (или просто ядро) — это такая функция, заданная на парах векторов, для которой существует спрямляющее пространство, в котором эта функция соответствует скалярному произведению.

Можно вообще не вводить новое пространство (достаточно знать, что оно существует), а сразу задать ядро K . Такой приём называется kernel trick.

Пример работы SVM с ядром приведён на рис. 2.8.

Приём с ядрами, как минимум, позволяет экономить память (не нужно хранить новые переменные). Кроме того, некоторые ядра соответствуют спрямляющим пространствам бесконечной размерности (и тут явно ввести переменные не получится).

Заметим, что далеко не любая симметричная функция, заданная на парах векторов, является ядром. Существует ряд требований, которым она должна удовлетворять.

2.5.6 Беспризнаковая классификация

Существует довольно большой класс задач, где объекты не имеют признакового описания. В качестве примеров можно привести музыкальные произведения, картины, тексты.

Конечно, для таких данных существуют способы построить признаковые пространства (например, с помощью нейросетей), но это требует большого объёма данных.

Однако в ряде задач мы можем каким-то образом (например, от экспертов) получить численные оценки похожести (или непохожести) объектов друг на друга. А непохожесть — это по смыслу «расстояние» (только аксиомы метрики могут не выполняться).

Вспомним, что самая известная метрика — Евклида — определяется на основе скалярного произведения.

Эти соображения наталкивают на идею построить по экспертным оценкам похожести метрику, а по метрике подобрать скалярное произведение, которого достаточно для применения SVM. Однако не любая непохожесть — метрика, и не в любом метрическом пространстве можно ввести скалярное произведение.

С другой стороны, именно скалярное произведение нам не требуется, поскольку достаточно функции ядра.

В результате применение метода SVM для задачи беспризнаковой классификации выглядит следующим образом.

Берётся матрица экспертных оценок «непохожести» объектов (если эксперт оценивал «похожесть», то превращаем её в разлиchie).

Эта матрица корректируется так, чтобы выполнялись требования к функции ядра.

На скорректированной матрице запускается SVM.

2.5.7 Задача безусловной оптимизации

Нужно отметить очень важный факт, а именно: задача 2.16 эквивалентна безусловной минимизации (по w, w_0) функции

$$\frac{w^2}{2} + C \sum_{i=1}^N \text{relu}(1 - y^i(x^i w - w_0)), \quad (2.19)$$

где

$$\text{relu}(z) = \begin{cases} z, & z > 0 \\ 0, & z \leq 0 \end{cases}.$$

Функция relu широко используется в нейросетях.

Может показаться, что решать задачу безусловной оптимизации должно быть проще, чем задачу квадратичного программирования. Зачастую это так, но не в данном случае. Функция relu имеет излом, что приведёт к затруднениям при использовании градиентных методов. Кроме того, такая формулировка не позволяет использовать приём с ядром.

Тем не менее, постановка 2.19 важна.

Во-первых, она показывает, что метод опорных векторов может быть сформулирован в форме минимизации потерь.

Во-вторых, такая форма позволяет увидеть сходство SVM и логистической регрессии. Для этого достаточно сравнить функции потерь.

Оба метода эквивалентны безусловной минимизации соответствующей функции потерь с регуляризацией.

На рис. 2.7 приведены функции потерь для SVM и логистической регрессии (из формулы 2.9). В качестве аргумента функций потерь взят так называемый отступ (margin), а именно выражение $y^i(x^i w - w_0)$. Видим, что функции разные по форме, но близки по значениям. Из этого следует, что и результаты этих методов должны быть (при прочих равных) похожи.

Такое сходство методов, в частности, даёт основания перенести из SVM вид регуляризатора, а именно положить в 2.9 $\kappa(w^2) = \frac{w^2}{2C}$.

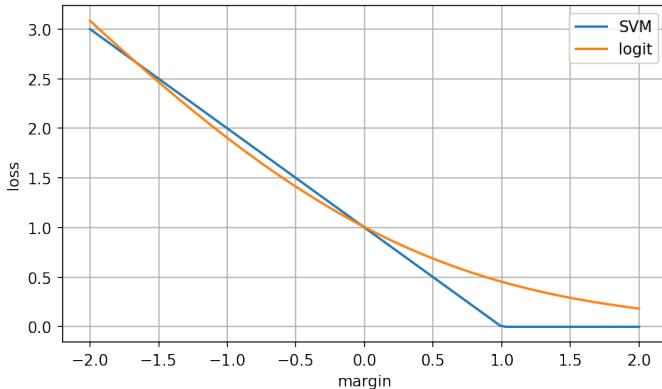


Рис. 2.7: Функции потерь для SVM и логистической регрессии.

2.5.8 Свойства метода

Подытожим основные свойства метода опорных векторов.

- Метод сводится к эффективно решаемой задаче квадратичного программирования.
- Разреженность (решение определяется только опорными векторами).
- Возможно обобщение введением функции ядра.
- Возможна беспризнаковая классификация.
- Низкая эффективность при сильно перекрывающихся распределениях классов.
- Неустойчив к выбросам. Целесообразна фильтрация как для логистической регрессии.

Существуют также вопросы, на которые пока нет точного ответа.

- Как соотносится SVM с метрическими методами.
- Является ли разреженность достоинством.

- Как выбирать ядро.
- Как применить kernel trick в логистической регрессии.

Пик популярности метода пришёлся на 90-е годы. Однако и в настоящее время метод остаётся вполне актуальным.

На большинстве задач метод показывает достаточно высокую точность. Чаще всего, эта точность несколько ниже, чем у бустинга, но несущественно. При этом в каких-то задачах SVM может оказаться даже лучшим по точности.

Однако, даже если точность SVM оказалась несколько ниже, чем у бустинга, может быть целесообразным использовать именно его, поскольку линейное решение имеет очень важное достоинство: простоту и интерпретируемость (к сожалению, это не относится к ядрам). В некоторых задачах это даже важнее точности.

2.6 Характеристика линейных методов

Междуд линейными методами мы установили достаточно неочевидную общность, т.е. сходство.

- Дискриминант Фишера не требует вероятностных предположений, но по результату почти совпадает с линейным дискриминантом, основанным на нормальности.
- Наивный байесовский классификатор является частным случаем логистической регрессии (дополненной заменой переменных).
- Метод SVM оказывается близок логистической регрессии ввиду сходства функций потерь.

В целом, линейные методы являются очень широким и важным классом методов машинного обучения и используются как самостоятельно, так и как фундамент для других методов (например, в нейросетях в основе лежит как раз линейная функция, а многие функции активации также заимствованы из линейных моделей).

2.7 Примеры кода

2.7.1 Визуализация решений

Первый пример демонстрирует визуализацию решающих функций, построенных разными методами на первых двух переменных таблицы Iris.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import datasets, svm, tree
from sklearn.discriminant_analysis import (
    LinearDiscriminantAnalysis,
    QuadraticDiscriminantAnalysis)
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import (
    GradientBoostingClassifier)

def make_meshgrid(x, y, padding = 1.0, N = 200):
    x_min = x.min() - padding
    x_max = x.max() + padding
    y_min = y.min() - padding
    y_max = y.max() + padding
    xx, yy = np.meshgrid(np.linspace(x_min, x_max, N),
                         np.linspace(y_min, y_max, N))
    return xx, yy

models = {
    'SVC with linear kernel':
        : svm.SVC(kernel='linear', C=1.0),
    'LinearSVC (linear kernel)':
        : svm.LinearSVC(C=1.0, max_iter=10000),
    'LogisticRegression':
        : LogisticRegression(),
    'LinearDiscriminantAnalysis':
        : LinearDiscriminantAnalysis(),
    'naive Gauss':
        : GaussianNB(),
    'SVC with RBF kernel':
        : svm.SVC(kernel='rbf', gamma=0.7, C=1.0),
    'SVC with polynomial (degree 2) kernel':
        : svm.SVC(kernel='poly', degree=2),
    'QuadraticDiscriminantAnalysis':
        : QuadraticDiscriminantAnalysis(),
```

```

'DecisionTreeClassifier',
: tree.DecisionTreeClassifier(),
'GradientBoosting',
: GradientBoostingClassifier(
    n_estimators=200, learning_rate=0.1,
    max_depth=1, random_state=0)
}

iris = datasets.load_iris()

X = iris.data[iris.target!=0, :2]
y = iris.target[iris.target!=0]

models = {title : clf.fit(X, y) for
          title, clf in models.items()}

plot_size = 15
cmap = plt.cm.winter
n_rows = (len(models)+1)//2
fig, sub = plt.subplots(n_rows, 2,
                       figsize=(plot_size, plot_size*n_rows//2))
plt.subplots_adjust(wspace=0.15, hspace=0.2)

X0, X1 = X[:, 0], X[:, 1]
xx, yy = make_meshgrid(X0, X1)

for (title, clf), ax in zip(models.items(),
                            sub.flatten()):
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    zz = Z.reshape(xx.shape)
    ax.contourf(xx, yy, zz, cmap=cmap, alpha=0.5)

    ax.scatter(X0, X1, c=y, cmap=cmap,
               s=20, edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel(iris['feature_names'][0])
    ax.set_ylabel(iris['feature_names'][1])
    ax.set_title(title)

plt.show()

```

Результат выполнения кода (частично) приведён на рис. 2.8.

Код может использоваться в качестве основы для дальнейших модификаций. В него можно легко добавить вызов других ме-

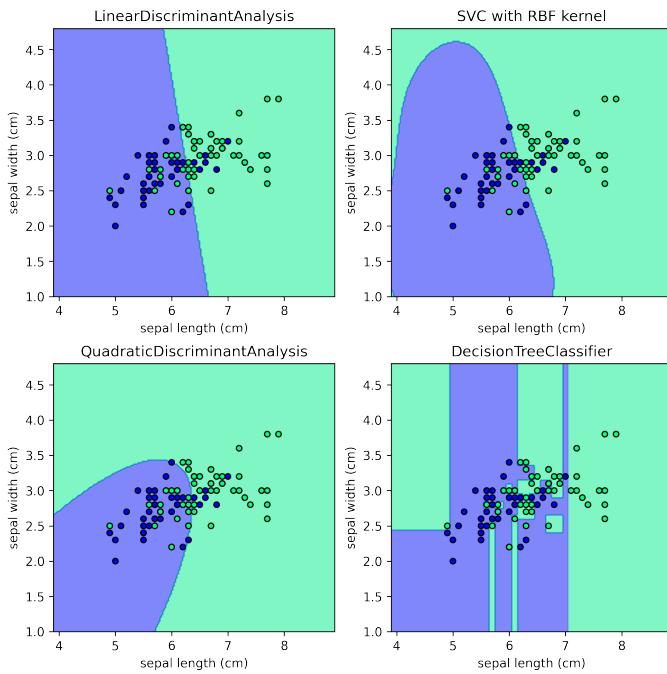


Рис. 2.8: Визуализация прогнозов разными методами.

дов, для чего достаточно модифицировать словарь `models`.

2.7.2 Кодирование категориальных переменных и наивный байесовский классификатор

Следующий код демонстрирует один из наиболее эффективных способов кодирование категориальных переменных, а именно, target encoding с логистической трансформацией.

На основе полученных значений наивный байесовский классификатор реализуется простым суммированием.

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression

# Загружаем данные из репозитория:
df = pd.read_csv(
    'https://archive.ics.uci.edu/ml/' +
    'machine-learning-databases/' +
    'mushroom/agaricus-lepiota.data', header=None)

# Первый столбец - целевая переменная
Y = (df[0]== 'p').astype(int)

# Вычисляем частоту первого класса:
p0 = Y.sum()/Y.shape[0]
z0 = np.log(p0) - np.log(1.0 - p0)
print('p0, z0:', p0, z0)

# Выполняем target encoding:
X = pd.DataFrame(index=df.index)
for i in df.columns[1:]:
    f = df.groupby(i)[0].value_counts(
        ).unstack().fillna(0)
    f[ 'prob' ] = (f[ 'p' ]+1)/(f[ 'p' ]+f[ 'e' ]+2)
    f[ 'logit' ] = np.log(f[ 'prob' ]) - np.log(1.0
                                                - f[ 'prob' ]) - z0
    X[ i ] = df[ i ].map(lambda x: f.at[x, 'logit'])

# Прогноз в предположении независимости:
P = X.sum(axis=1) + z0
P = 1/(1+np.exp(-P))
print('Naive Bayes accuracy:',
```

```
((P>0.5)==(Y==1)).sum() / Y.shape[0])  
  
#Логистическая регрессия:  
clf = LogisticRegression().fit(X,Y)  
P1 = clf.predict(X)  
print('LogisticRegression accuracy:',  
((P1>0.5)==(Y==1)).sum() / Y.shape[0])
```

В результате выполнения кода будет выведено.

```
p0, z0: 0.482028557 -0.07191675  
Naive Bayes accuracy: 0.956548498  
LogisticRegression accuracy: 1.0
```

Автор настоятельно рекомендует запустить все предложенные примеры и понять работу кода, в том числе внося в него изменения и наблюдая результат.

Глава 3

Логические методы классификации

Логические методы — широко используемый класс методов.

Основные варианты:

- поиск закономерностей, решающие списки,
- решающие деревья.

В настоящее время из логических методов наиболее известны деревья решений, которые используются в том числе как базовый классификатор для бустинга и случайного леса. Но для полноты картины рассмотрение стоит начать с понятия логической закономерности.

3.1 Логические закономерности

3.1.1 Понятие закономерности

Слово «закономерность» обычно используется как антоним слова «случайность».

Однако в данной работе мы будем термин «случайность» использовать исключительно в определённом математически строгом смысле, а именно: будем называть случайнym то, чему мы

приписали вероятность. Поэтому «закономерность» будем полагать скорее антонимом слову «хаотичность».

Итак, под закономерностью в широком смысле будем понимать какую-то особенность в данных, которая обладает статистической устойчивостью.

При этом мы предполагаем, что закономерности нам понадобятся в конечном итоге для прогнозирования класса объекта, поэтому нас интересуют такие особенности в данных, которые можно использовать для прогнозирования.

В результате закономерностью будем называть заданной формы область в пространстве переменных, в которой объектов одного класса «много», а других классов «мало» (смысл этих понятий уточним ниже).

Первым делом нужно выбрать форму областей. Очевидно, что форма должна быть простой, поскольку областью достаточно сложной формы можно отделить любое подмножество точек выборки.

Самая простая форма — многомерный интервал (гиперпараллелепипед). Исторически рассматривались и другие формы (в частности, сферы), но мы ограничимся интервалами.

Наверное, уже напрашивается вопрос: где здесь «логика», т.е. почему мы называем закономерности логическими, а не к примеру «геометрическими», раз уж речь идёт о гиперпараллелепипедах.

Дело в том, попадание точки в область удобно описывать в терминах предикатов.

Собственно, предикат — это такая функция, которая в качестве аргумента принимает точку, а в качестве результата возвращает значение «истина», если точка попала в заданную область, и «ложь» — иначе.

Предикаты удобны тем, что для них можно использовать логические операции (например, конъюнкцию), а также предикаты удобно использовать в узлах дерева.

Для примера приведём логические закономерности для задачи Iris 3.1.1.

На рисунке приведено по одной закономерности для каждого класса. Если с первым классом вопросов не возникает, то интервал для второго класса не захватывает два объекта этого класса.

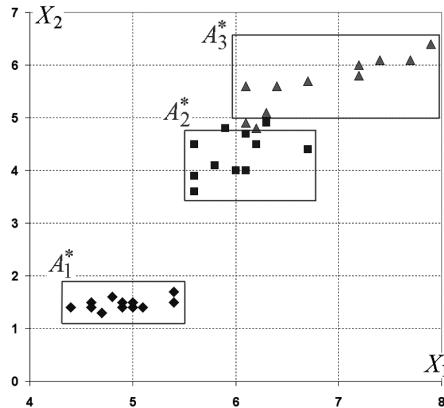


Рис. 3.1: Логические закономерности для задачи Iris

Мы можем интервал расширить, включив эти объекты, но тогда попадут также точки третьего класса. Вопрос, какой вариант лучше. Чтобы на него ответить, нужно формализовать критерий качества.

Пусть X – пространство значений прогнозирующих переменных, $Y = \{1, 2, \dots\}$ – прогнозируемая переменная, $\varphi : X \rightarrow \{0, 1\}$ – предикат.

Классы могут нумероваться (обозначаться) произвольным образом, но будем считать, что среди них есть класс, обозначенный как первый.

Поскольку закономерности будем искать для каждого класса независимо, нам достаточно рассмотреть эту процедуру для первого класса.

Пусть $S = ((x^i, y^i) \mid i = \overline{1, N})$ – выборка объектов, из которых M принадлежит 1-му классу, n – число точек (выборки), на которых предикат φ истинный, m – сколько из них 1-го класса.

Закономерность тем «лучше» («сильнее»), чем больше она включает объектов своего класса и чем меньше чужих. Это можно записать как

$$a = \frac{m}{M} \rightarrow \max, \quad b = \frac{n - m}{n} \rightarrow \min.$$

Величины и обе характеризуют качество закономерности. Необ-

ходимо свести их в одно выражение, т.е. задать единый критерий качества. Это можно сделать множеством способов, например

$$\frac{a}{b} \rightarrow \max, \quad a - b \rightarrow \max, \quad \sqrt{a} - \sqrt{b} \rightarrow \max.$$

Нужно понять, какое из выражений более «правильное», и можно ли получить подобное выражение не эвристически, а из рациональных рассуждений.

3.1.2 Критерии качества закономерности

Естественно считать закономерность тем более сильной, чем менее она случайна. Этот тезис интуитивно вполне понятен, но мы только что договорились не использовать термин «случайность» в том смысле, который тут вкладывается. Изменим формулировку, заодно сделав её строгой.

Будем считать закономерность тем более сильной, чем в большей степени (на меньшем уровне значимости) она отвергает «нулевую гипотезу».

В задаче классификации естественным вариантом нулевой гипотезы является предположение, что все классы имеют одинаковое распределение. Эквивалентная формулировка: вероятность попадания объекта в область не зависит от класса, к которому он принадлежит.

Идея статистического критерия (согласия) для проверки гипотез заключается в следующем. Выбирается некоторое событие, вероятность которого при выполнении гипотезу очень низка (порог, ниже которого вероятность считается низкой, называется уровнем значимости). Если это событие всё же происходит, то гипотеза отвергается.

Пусть в некоторую область пространства попало n объектов. Какова вероятность, что среди них окажется m объектов первого класса?

Можно привести практическую иллюстрацию. Выпущена лотерея на N билетов, из которых M выигрышных. Купили n билетов. Какова вероятность, что из них будет m выигрышных?

Это простая типовая задача по теории вероятностей, ответ в которой выражается формулой гипергеометрического распределения.

Вероятность при отборе n объектов получить m из них 1-го класса равна

$$P(m) = \frac{C_M^m C_{N-M}^{n-m}}{C_N^n} = \frac{C_n^m C_{N-n}^{M-m}}{C_N^M}.$$

Однако для выражения качества закономерности следует использовать не $P(m)$, а сумму

$$U(m) = \sum_{m'=m}^M P(m'),$$

которая есть вероятность того, что в области окажется не менее m объектов 1-го класса.

Причина, по которой нужно суммировать $P(m)$, объясняется в курсах математической статистики в теме о критериях согласия. В данном пособии аналогичное объяснение встретится в разделе с оценкой доверительного интервала по контрольной выборке, поэтому если данный момент сейчас непонятен, к нему можно вернуться после прочтения указанного раздела.

Содержательно $U(m)$ есть достигаемый уровень значимости для проверки нулевой гипотезы. Чем он ниже, тем «сильнее» отвергается гипотеза.

Заметим, что $U(m)$ не зависит от сложности закономерности и размерности пространства, поскольку посчитана для априорно выбранной закономерности.

Если мы ищем закономерность, целенаправленно минимизируя $U(m)$, то полученный минимум уже не годится в качестве достигаемого уровня значимости для проверки нулевой гипотезы. Требуется поправка, которая уже зависит от сложности предиката. Для оценки этого эффекта потребуются рассуждения в духе принципа равномерной сходимости.

Вероятность $P(m)$ можно вычислять приближённо.

Введём обозначение

$$H(p) = -p \ln p - (1-p) \ln(1-p).$$

Данная величина есть энтропия (часто используют логарифм по основанию 2, но для удобства возьмём натуральный).

Вспомним формулу Стирлинга:

$$\ln(k!) \approx k \ln k - k + \frac{1}{2} \ln(2k\pi) + \frac{1}{12k} - \frac{1}{360k^2}.$$

Воспользовавшись первыми двумя членами данного разложения, оценим логарифм биномиального коэффициента

$$\begin{aligned}\ln C_N^n &= \ln \left(\frac{N!}{n!(N-n)!} \right) \approx \\ &\quad N \ln N - N - (n \ln n - n) - \\ &\quad ((N-n) \ln(N-n) - (N-n)) = \\ &\quad -n(\ln n - \ln N) - (N-n)(\ln(N-n) - \ln N) = \\ &\quad NH\left(\frac{n}{N}\right).\end{aligned}$$

Оказывается, биномиальный коэффициент просто выражается через энтропию. Заметим, что вторые члены приближения попросту сократились.

Применим это выражения для оценки вероятности закономерности. Получим

$$\ln P(m) \approx -NG,$$

где

$$G = H\left(\frac{M}{N}\right) - \frac{n}{N} \cdot H\left(\frac{m}{n}\right) - \frac{N-n}{N} \cdot H\left(\frac{M-m}{N-n}\right).$$

Здесь G – так называемый информационный критерий или количество информации в закономерности. Эта величина есть разность между изначальной энтропией системы и её энтропией после выделения заданной области.

3.1.3 Решающие списки

Когда выбран критерий качества, можно создавать алгоритм поиска закономерностей. Основная техническая проблема заключается в большом пространстве поиска, практически исключающем полный перебор. Напрашивается использование направленного перебора.

Схема направленного поиска закономерностей («жадный» алгоритм):

- дискретизация признаков (границы между проекциями точек выборки);
- интервальные предикаты;
- конъюнкции элементарных предикатов.

Наиболее известные алгоритмы такого типа: КОРА, ТЭМП.

В результате работы этих алгоритмов мы получаем большой список закономерностей со значениями критерия качества. Список целесообразно упорядочить по качеству.

На основе списка закономерностей можно решать задачу классификации. Решение принимается по первой закономерности, которой удовлетворяет объект, т.е. приписывается тот номер класса, к которому относится закономерность. Можно применять голосование по нескольким самым информативным закономерностям.

Помимо возможности решения задачи классификации логические закономерности имеют самостоятельную ценность, поскольку отражают особенности данных, что полезно для их анализа.

3.2 Решающие деревья

При независимом поиске закономерностей они оказываются сильно пересекающимися. Если основная цель — решение задачи классификации, то имеет смысл искать закономерности так, чтобы их набор сразу давал разбиение пространства переменных.

Напомним, что разбиением называется представление множества в виде объединения непересекающихся подмножеств.

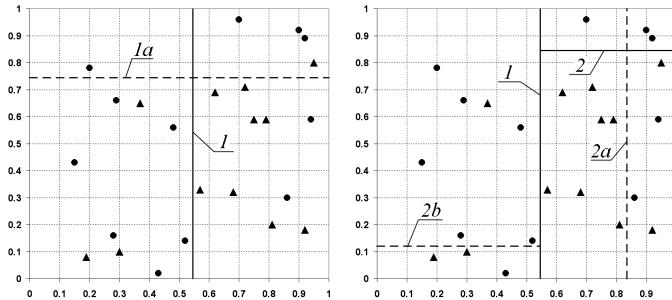


Рис. 3.2: Построение разбиения деревом решений

Дерево решений строит иерархическое разбиение пространства переменных.

3.2.1 Понятие дерева решений

Решающие деревья являются одним из наиболее часто используемых инструментов для анализа данных. К достоинствам решающих деревьев относятся их наглядность, применимость к широкому кругу задач, возможность варьировать сложность решений, подстраиваясь под различный объём выборки, способность автоматически выбирать информативные переменные.

Методы, основанные на деревьях решений, применяются почти во всех задачах анализа данных. В первую очередь, рассмотрим задачу классификации.

На рис. 3.2.2 приведена модельная (т. е. созданная искусственно, а не взятая из реальной задачи) выборка, представленная 24-я точками двух классов в двумерном пространстве. Обе переменные количественные и область значений каждой представляет собой интервала $[0, 1]$. Треугольные маркеры соответствуют первому, круглые — второму классу.

3.2.2 Построение дерева решений

Дерево представляет собой последовательное разбиение пространства значений на области.

Идея метода, основанного на решающих деревьях, заключается в последовательном разбиении пространства значений переменных на области E_1, \dots, E_L и приписывании каждой полученной области решения в виде номера класса.

Под последовательным разбиением понимается то, что сначала всё пространство $X = X_1 \times X_2$ разбивается на две области $E_1^{(1)}$ и $E_2^{(1)}$, из которых затем выбирается область, которая, в свою очередь, разбивается на две, в результате чего получается три области $E_1^{(2)}, E_2^{(2)}, E_3^{(2)}$, и так далее, пока через $L - 1$ таких шагов не получится L областей.

Заметим, что способов подобного разбиения пространства X на L областей очень много. Естественно выбрать разбиение, минимизирующее число ошибочно классифицируемых объектов выборки. Однако число различимых на выборке объёма N разбиений в случае n количественных переменных составляет порядка $(nN)^L$, поэтому полный перебор всех разбиений в реальных задачах обычно невозможен.

При этом задача поиска разбиения, точно классифицирующего обучающую выборку и имеющего минимальное число областей, является NP-полной. Это значит, что не следует рассчитывать на появление имеющих низкую трудоёмкость точных алгоритмов нахождения наилучшего разбиения. В этой ситуации используют эвристические алгоритмы, простейшим вариантом которых является так называемый «жадный» алгоритм.

Рассмотрим работу «жадного» алгоритма на примере построения разбиения выборки из рис. 3.2.2.

На первом шаге алгоритм перебирает все варианты разбиения пространства X на две области. Разбиение можно проводить как по переменной X_1 , так и по X_2 .

Переберём сначала варианты разбиения по переменной X_1 .

Хотя число вариантов разбиения области значений количественной переменной на два интервала бесконечно (континуально), нас интересуют только варианты, различимые на обучающей выборке, поэтому будем рассматривать только границы, расположенные посередине между проекциями выборочных точек.

Каждая граница s разбивает пространство X на области $E_1^{(1)} =$

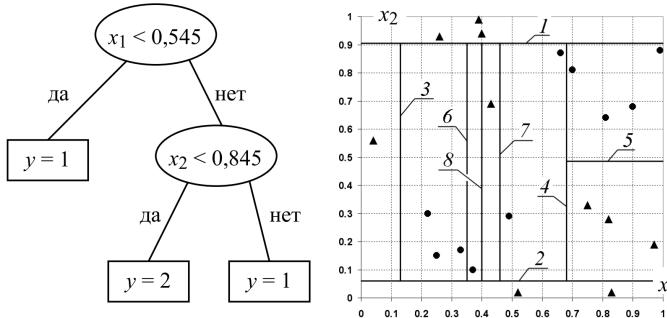


Рис. 3.3: а) Дерево решений. б) Неудачное разбиение для задачи «исключающего или».

$[0, s] \times [0, 1]$ и $E_2^{(1)} = [s, 1] \times [0, 1]$. Обозначим за $N_l^1(s)$ число точек первого, а $N_l^2(s)$ – число точек второго класса в области $E_l^{(1)}$.

Естественно, что в каждой получившейся области решающая функция припишет тот класс, выборочных объектов которого в этой области больше. Тогда оставшиеся объекты будут классифицированы ошибочно. Таким образом, число ошибочно классифицированных объектов составит

$$\tilde{N}(s) = \min(N_1^1(s), N_1^2(s)) + \min(N_2^1(s), N_2^2(s)).$$

Перебрав все варианты разбиения пространства по переменной X_1 , видим, что наименьшее число ошибок (восемь) получается при $s = 0,545$. Соответствующая граница показана на рис. 3.2.2 цифровой 1.

Аналогичным образом следует перебрать все варианты разбиения по переменной X_2 . Наименьшее число ошибок (девять) будет достигнуто для границы, обозначенной на рис. 3.2.2 как 1а.

Поскольку разбиение по переменной X_1 даёт меньшее число ошибок, окончательно выбираем разбиение с границей 1, т.е. $E_1^{(1)} = [0; 0,545] \times [0, 1]$, $E_2^{(1)} = [0,545; 1] \times [0, 1]$.

На втором шаге алгоритма аналогичным образом перебираем варианты разбиений областей $E_1^{(1)}$ и $E_2^{(1)}$.

В результате перебора находим (см. рис.3.2.2), что граница 2б уменьшает общее число ошибок на 1, граница 2а уменьшает общее

число ошибок на 2, а граница 2 уменьшает общее число ошибок на 3. В результате выбираем вариант 2 и получаем разбиение на области

$$\begin{aligned} E_1^{(2)} &= [0; 0,545) \times [0; 1], \\ E_2^{(2)} &= [0,545; 1] \times [0; 0,845), \\ E_3^{(2)} &= [0,545; 1] \times [0,845; 1]. \end{aligned}$$

Решающая функция $f(x)$ принимает значение 1 при $x \in E_1^{(2)} \cup E_3^{(2)}$ и значение 2 при $x \in E_2^{(2)}$.

На рис. 3.2.2 слева эта же решающая функция представлена в форме дерева решений.

На рис. 3.2.2 справа приведён пример другой выборки. Легко заметить, что данная выборка безошибочно классифицируется при разбиении

$$\begin{aligned} E_1 &= [0; 0,5) \times [0; 0,5), \quad E_2 = [0,5; 1] \times [0; 0,5), \\ E_3 &= [0; 0,5) \times [0,5; 1], \quad E_4 = [0,5; 1] \times [0,5; 1], \end{aligned}$$

При этом «жадный» алгоритм достигает безошибочной классификации только при разбиении на 9 областей (показаны на рис. 3.2.2, цифрами отражён порядок проведения границ). Заметим, что некоторые проведённые границы вообще не уменьшают число ошибок и проводились так, чтобы разделять область с наибольшим числом объектов на подобласти, разделяя объекты примерно поровну.

Последний пример демонстрирует, что «жадный» алгоритм в общем случае находит далеко не лучшее решение. Существуют многочисленные усовершенствованные варианты алгоритма направленного поиска дерева решений, однако до настоящего времени проблема построения наиболее эффективного алгоритма остаётся открытой.

Одно из важнейших достоинств решающих состоит заключается в том, что основанные на них методы построения решающих функций очень легко переносятся на случай разнотипного пространства переменных. Если для вещественных переменных в ка-

чество областей разбиений брались интервалы, то для номинальных переменных в разбиениях участвуют любые подмножества значений.

3.2.3 Критерии ветвления

При построении дерева требуется оценивать качество получаемых вариантов. Если мы наращиваем дерево последовательно, то на каждом шаге требуется сравнивать варианты, которые отличаются только одним нетерминальным узлом и выходящими из него конечными вершинами. Поэтому технически критерий качества можно пересчитывать только для изменяющихся вершин. Иногда критерий выписывают в форме разности качества разных вариантов, однако для понимания проще рассматривать критерий качества всего дерева.

Критерий качества дерева задаётся как функция от количества объектов каждого класса в конечных вершинах.

$$K(\lambda) = \sum_{t=1}^T N^t L(\nu_1^t, \dots, \nu_k^t),$$

где N^t – число объектов в конечной вершине t , $(\nu_1^t, \dots, \nu_k^t)$ – относительные частоты классов в вершине t , T – число конечных вершин дерева.

Критерии $L(\dots)$ оценки дерева (для классификации):

- число ошибок классификации (для детерминированного прогноза),
- критерий Джини (Gini impurity) – это по сути число ошибок для вероятностного прогноза,
- информационный (log loss, на основе функции правдоподобия для вероятностного прогноза).

Выражение для критерия числа ошибок

$$L_E(\nu_1^t, \dots, \nu_k^t) = 1 - \max_{\omega}(\nu_{\omega}^t).$$

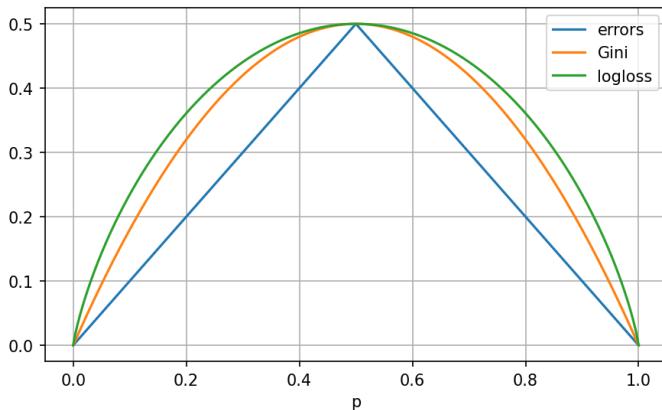


Рис. 3.4: Сравнение критериев ветвления.

Эта величина есть доля ошибочно классифицированных объектов, если всей вершине приписан наиболее частый (в ней) класс.

Выражение для критерия Джини

$$L_G(\nu_1^t, \dots, \nu_k^t) = 1 - \sum_{\omega=1}^k (\nu_\omega^t)^2.$$

Может показаться странным, что частота возводится в квадрат. На самом деле критерий имеет простой смысл: это вероятность ошибочной классификации объекта, если класс будем приписывать случайно с вероятностями, равными относительным частотам.

Выражение для критерия log loss

$$L_L(\nu_1^t, \dots, \nu_k^t) = - \sum_{\omega=1}^k \nu_\omega^t \ln(\nu_\omega^t).$$

Это критерий максимума правдоподобия для вероятностного прогноза в соответствии с частотами.

Первый критерий (доля ошибок) работает плохо, остальные сопоставимо.

На рис. 3.2.3 можно увидеть, что Gini impurity и log loss отличаются, но не слишком сильно.

3.2.4 Алгоритмы

Варианты алгоритма построения дерева:

- жадный (без учёта будущих ветвлений),
- рекурсивный (предикат в узле выбирается с учётом ветвления на нижнем уровне, решает задачу XOR),
- неограниченный (строит дерево, затем его оптимизирует всё доступное время).

Сейчас на практике используется «жадный» алгоритм.

3.3 Выводы

Недостатки логических методов:

- невозможность «гладких» решений,
- много эвристик (настраиваемых параметров),
- вычислительная трудоёмкость нахождения точных решений,
- неизвестен лучший критерий ветвления.

Актуальность:

- возможность получить интерпретируемое решение,
- используется в составе бустинга.

Свойства:

- работа в разнотипном пространстве (однако большинство реализаций это не поддерживают),
- работа с пропусками,
- интерпретируемость решений.

Бустинг на деревьях — один из лучших методов.

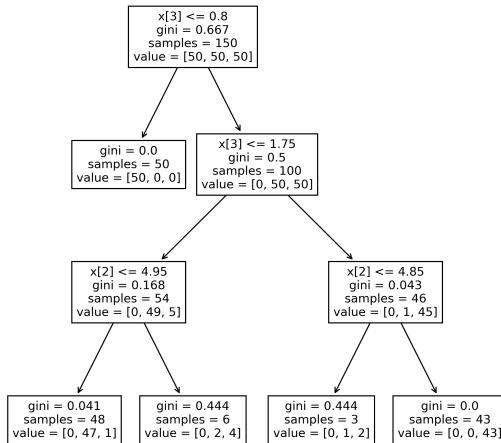


Рис. 3.5: Дерево решений для задачи Iris.

3.4 Примеры кода

Приведём пример кода для построения дерева и его визуализации.

В примере также показана возможность сохранения рисунка в файл.

```

import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn import tree

clf = tree.DecisionTreeClassifier(max_depth=3,
                                  random_state=0)
iris = load_iris()
clf.fit(iris.data, iris.target)

plt.figure(figsize=(8,8))
tree.plot_tree(clf)
plt.savefig('tree_iris.png', dpi=300,
            bbox_inches='tight')
plt.show()

```

Результат выполнения кода приведён на рис. 3.5. Размер (глубина) дерева принудительно ограничен с помощью соответствующего параметра.

Глава 4

Ансамблевые методы

4.1 Композиции решающих функций

Основные понятия $X = R^n$ – пространство значений прогнозирующих переменных,
 $Y = \{-1, 1\}$ – прогнозируемая переменная,
 $D = X \times Y$.

Решающая функция (алгоритм классификации)

$$f : X \rightarrow Y.$$

$S = ((x^i, y^i) \in D \mid i = \overline{1, N})$ – случайная независимая выборка,
 $S \in D^N$.

$Q: D^N \rightarrow \Phi$ – метод построения решающих функций,
 Φ – заданный класс решающих функций.

Композиции классификаторов

Обобщение решающей функции: $\lambda: X \rightarrow [0, +\infty)$ – вводится
пространство оценок.

Пусть имеются T решающих функций $\lambda_1(x), \dots, \lambda_T(x)$.

Композиция есть решение в виде

$$\lambda(x) = C(\lambda_1(x), \dots, \lambda_T(x)),$$

где $C(\cdot, \dots, \cdot)$ – монотонна по всем аргументам.

Функции $\lambda_t(x)$ принимают значения из пространства оценок, значения функции $\lambda(x)$ – из множества Y .

Линейные композиции

Линейная композиция

$$\lambda(x) = \text{sign} \left(\sum_{t=1}^T \alpha_t \lambda_t(x) \right).$$

Если $\alpha_t \geq 0$, то речь идёт о выпуклых композициях (чаще всего рассматривают именно их).

Методы построения композиций

- Бустинг (AdaBoost, градиентный бустинг)
- Бэггинг и метод случайных подпространств (случайный лес)
- Нейронные сети
- Голосование (блэндинг)
- Стэкинг (решения в качестве новых переменных)

Смеси алгоритмов – если α_t зависит от x , идея областей компетентности.

4.2 Независимые подвыборки

Предположим, что у нас есть некоторый метод классификации (или в общем случае, построения решающих функций), который назовём базовым методом.

Мы хотим на его основе построить ансамбль из решений. Для этого нужно каким-то образом добиться, чтобы один и тот же метод выдавал разные решения.

Относительно многие реализации методов при поиске решения используют случайность. Для таких реализаций можно добиться разных решений, задавая разные значения параметра базы генератора псевдослучайных чисел.

Но такой подход, во-первых, не для всех алгоритмов возможен, а во-вторых, получаемые таким образом решения для нас «недостаточно различны».

Кроме того, формально под методом построения решающих функций мы понимаем отображение из множества выборок во множество решений. В этом случае получить одним методом разные решения можно, только подав на вход разные выборки. При этом обучающая выборка у нас изначально одна.

Простейший и совершенно напрашивающийся способ получить из одной выборки несколько — это брать случайные подвыборки. При этом (если представлять выборку в виде таблицы данных) можно выбирать как строки (объекты), так и столбцы (признаки).

Оба этих варианта используются на практике (как правило, совместно).

4.2.1 Бэггинг

Выбирая случайные подмножества строк и столбцов таблицы данных, можно получать различные решающие функции. Но что если мы захотим получить «ещё более различные» базовые решения?

Для этих целей придумана идея бутстрэп-выборок, а весь подход назван бэггингом (bagging — сокращение от bootstrap aggregation).

Чтобы добиться большего различия решений мы будем не просто брать подвыборку, но ещё и искажать её. При этом искажения будут очень умеренными: мы всего лишь повторим некоторые строки два или более раз. Фактически мы этим просто приписываем объектам некоторый случайный (целочисленный) вес.

Но как именно реализовать подобное искажение подвыборок? Для этого предложена изящная идея бутстрэп-выборок.

Идея проста: формировать новую выборку на основе исходной как выборку с повторениями, причём размер новой выборки задаётся равным исходной.

Когда говорят о случайной подвыборке, то подразумевают, что подвыборка формируется без повторений: случайный объект ис-

ходной выборки мы как бы переносим в новую выборку, и в следующий раз он нам попасться не может.

Формируя выборку с повторениями, мы как бы копируем объект в новую выборку — и он может быть выбран много раз.

Если размер формируемой выборки равен исходной, то в среднем $(1 - \frac{1}{N})^N \approx e^{-1} \approx 37\%$ объектов в ней не попадут (на их месте будут копии других объектов).

Таким образом, идея бэггинга очень простая. Осталось лишь добавить, что она... как ни странно, работает.

4.2.2 Случайный лес

Метод случайного леса (random forest) строит ансамбль деревьев путём простого усреднения решений, построенных по bootstrap-выборкам на случайному подмножестве переменных.

Увеличение числа деревьев в методе случайного леса не увеличивает (в отличие от бустинга) эффективную сложность итогового решения и не может привести к падению качества (на контрольной выборке).

Таким образом, увеличивая число деревьев, можно только улучшить решение, однако использовать слишком большое число деревьев (больше сотен) обычно не имеет смысла, поскольку качество достаточно быстро стабилизируется на одном уровне.

Случайный лес использует привычные реализации алгоритма построения деревьев.

Соответственно он требует подбора (оптимизации) ровно тех же параметров, которые подбираются при построении дерева решений.

В первую очередь, требуется подбирать размер (глубину) дерева. Но если при построении одного дерева случаи, когда наилучшим выбором оказывались деревья неограниченной глубины, можно считать экзотикой, то для случайного леса такой вариант довольно реалистичен (но не настолько, чтобы пренебрегать оптимизацией глубины).

4.2.3 Свойства

Случайный лес обычно даёт достаточно хорошее решение, однако автору не известен ни один случай, когда (оптимально настроенный) случайный лес превзошёл бы по качеству (оптимально настроенный) бустинг (это не значит, что таких случаев нет).

Как правило, случайный лес даёт качество выше, чем одно дерево. Это кажется естественным, но по факту вовсе не очевидно, что так и должно быть.

Очевидно, что ансамбль независимых деревьев должен работать лучше, чем одно такое дерево.

Но дело в том, что в случайном лесе деревья «не такие», как при обычном построении дерева. Обычно мы дерево строим по всем переменным и по всей (неискажённой бутстрэпом) выборке. Случайный же лес строит искусственно «ослабленные» деревья.

Можно доказать (через разложение ошибки на смещение и разброс), что случайный лес как ансамбль ослабленных деревьев должен работать лучше, чем одно ослабленное дерево. Но ниоткуда (кроме интуиции) не следует, что ансамбль ослабленных деревьев должен работать лучше, чем одно «полноценное» дерево.

Следует также заметить, что строго говоря, деревья случайного леса не являются независимыми, поскольку строятся на подвыборках одной и той же выборки. Отсюда, в частности, следует, что дисперсию прогноза невозможно уменьшать до нуля (в пределе), увеличивая число деревьев.

4.3 Бустинг

Как и в предыдущем разделе, мы хотим получать одним базовым методом различные решения путём построения новых выборок на основе исходной.

Но теперь новые выборки будут получаться за счёт изменения весов объектов (а для градиентного бустинга будут также корректироваться значения целевой переменной).

Идея бустинга заключается в том, чтобы каждый последующий базовый классификатор исправлял ошибки предыдущих.

Вторым принципом бустинга является то, что мы не пытаемся улучшить ранее построенные классификаторы, т.е. если базовая решающая функция построена, то она включается в композицию и в дальнейшем не изменяется.

Рассмотрим модельный пример. Пусть есть некоторая выборка и базовый метод построил по ней линейный классификатор как на рис. 4.1.

Ошибочно классифицированные объекты выделены крупными маркерами.

Представим теперь, что мы хотим, чтобы следующий классификатор исправил эти ошибки. Простейший способ этого добиться — это оставить только эти ошибочно классифицированные объекты, а остальные просто убрать из выборки. Легко представить, к чему это приведёт: второй классификатор разобъёт пространство на примерно те же полуплоскости, но метки классов припишет им противоположным образом. Если теперь усреднить прогнозы от двух полученных решений, то... ничего хорошего не получится: прогноз будет везде нулевым (или совпадёт с одним из базовых решений, если веса при усреднении возьмём неравными).

Данные рассуждения призваны показать, что несмотря на очевидность идеи бустинга, реализовывать её нужно аккуратно.

4.3.1 Алгоритм AdaBoost

Увеличиваем веса объектов, на которых допущена ошибка.

Данный алгоритм является исторически первым и самым простым вариантом бустинга.

Изложим AdaBoost как собственно алгоритм классификации.

В методе AdaBoost решение строится в виде композиции

$$\lambda(x) = \text{sign}(\beta_T(x)), \quad \beta_T(x) = \sum_{t=1}^T \alpha_t \lambda_t(x),$$

где базовые классификаторы $\lambda_t(x)$ и их веса α_t находятся следующим образом.

Первый базовый классификатор строится базовым методом на основе исходной выборки, объектам которой приписаны начальные веса $w^1 = (w_1^1, \dots, w_N^1)$.

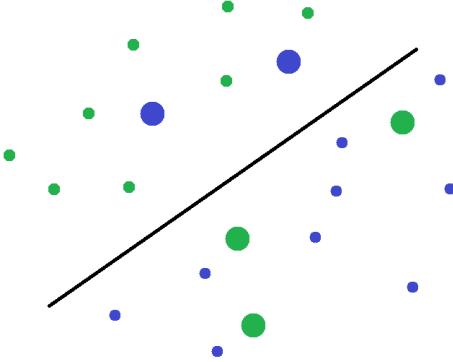


Рис. 4.1: Идея бустинга.

Заметим, что мы будем задавать начальные веса объектом в соответствии с выбранным распределением, но в стандартном варианте метода начальные веса выбираются одинаковыми, т.е. $w_i^1 = \frac{1}{N}$.

Пересчёт весов

Вес построенного базового классификатора в композиции определяется по формуле

$$\alpha_t = \frac{1}{2} \ln \frac{W^+(S, w^t, \lambda_t)}{W^-(S, w^t, \lambda_t)},$$

где

$$W^+(S, w, \lambda) = \sum_{i=1}^N w_i \cdot I(y^i = \lambda(x^i)),$$

$$W^-(S, w, \lambda) = \sum_{i=1}^N w_i \cdot I(y^i = -\lambda(x^i)).$$

Итерационный процесс

Следующие базовые классификаторы строятся тем же базовым методом по выборке, веса объектов в которой вычисляются по формулам

$$w_i^{t+1} = \frac{\bar{w}_i^{t+1}}{\sum_{i=1}^N \bar{w}_i^{t+1}}, \quad \bar{w}_i^{t+1} = w_i^t \cdot e^{-\alpha_t y^i \lambda_t(x^i)}.$$

Веса правильно классифицированных объектов умножаются на $e^{-\alpha_t}$, а веса неправильно классифицированных объектов умножаются на e^{α_t} .

Как и в классическом варианте алгоритма, мы здесь использовали перенормировку весов на каждом шаге. На самом деле, её можно было не делать, т.к. на решение она никак не влияет. Единственный смысл этой нормировки — избежать очень больших и очень маленьких значений веса, которые теоретически могут выйти за границы диапазона машинного представления вещественных чисел.

4.3.2 Свойства

Если проанализировать результаты соревнований (конкурсов) по анализу данных, то можно сделать вывод, что на задачах с табличными данными из всех методов именно бустинг чаще всего даёт самые точные решения.

То что бустинг решает задачу точнее, чем одно дерево, кажется естественным, но по факту вовсе не очевидно, что так и должно быть.

Естественно, что ансамбль деревьев работает лучше, чем одно дерево из ансамбля.

Но в бустинге деревья «не такие», как при обычном построении дерева. Обычное дерево строится по исходной выборке.

Бустинг строит деревья по изменённой выборке, и эти деревья вообще не предназначены для использования в качестве самостоятельного решения, поскольку обучались корректировать ошибки.

Можно доказать (через разложение ошибки на смещение и разброс), что бустинг как ансамбль специфических деревьев должен работать лучше, чем одно дерево из ансамбля. Но ниоткуда (кроме интуиции) не следует, что ансамбль специфических деревьев должен работать лучше, чем одно «полноценное» дерево.

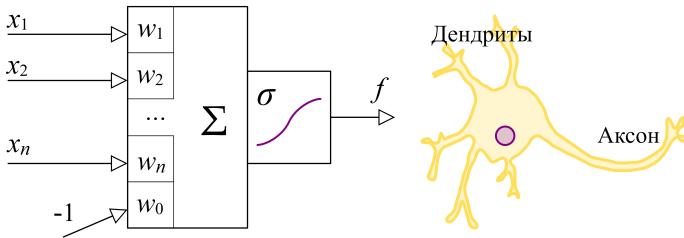


Рис. 4.2: Схемы искусственного и реального нейронов.

4.4 Нейронные сети

Под нейронной сетью (прямого распространения) понимается решающая функция в виде суперпозиции произвольного количества линейных функций многих переменных и нелинейных функций одной переменной.

Без ограничения общности можно считать, что линейные и нелинейные функции в суперпозиции строго чередуются (по уровням), поскольку суперпозиция произвольного числа линейных функций является линейной функцией (аналогичным образом, идущие подряд нелинейные функции можно считать одной функцией).

Последовательно применённые линейную и нелинейную функции будем называть (искусственным) нейроном.

4.4.1 Схема нейрона

Схемы искусственного и реального нейронов приведены на рис. 4.2. Математически нейрон представляет собой функцию вида

$$f(x) = \sigma(wx - w_0),$$

где x – вектор входных переменных, w – вектор весов, а $\sigma(\cdot)$ – так называемая функция активации.

Несмотря на простоту, математический нейрон, пожалуй, всё же является некоторой (разумеется, грубой) моделью нейрона реального. Действительно, веса входов могут соответствовать раз-

ной «проводимости» дендритов, а функция активации некоторым образом соответствует процессу возбуждения нейрона, в результате которого сигнал передаётся по аксону.

Ранее обозначение σ мы использовали для логистической функции. В дальнейшем будем этого придерживаться, но в контексте нейронных сетей σ будет обозначать функцию активации.

Теоретически функция активации может быть любой, но на практике используются только монотонные. Из монотонных без ограничения общности можно оставить только монотонно неубывающие, поскольку произвольный знак обеспечивается линейными коэффициентами.

4.4.2 Аппроксимирующая способность сети

Для начала выясним, как нейросетью реализовать булевы функции.

Начнём с простейших. Пусть x_1, x_2, x – булевые переменные, тогда

$$\begin{aligned}x_1 \wedge x_2 &= I(x_1 + x_2 - \frac{3}{2} > 0), \\x_1 \vee x_2 &= I(x_1 + x_2 - \frac{1}{2} > 0), \\\neg x &= I(-x + \frac{1}{2} > 0).\end{aligned}$$

Теперь мы можем прийти к очевидному заключению, что двухслойная сеть может представить произвольную булеву функцию. Это следует из того, что любая функция может быть представлена дизъюнктивной или конъюнктивной нормальной формой, а дизъюнкция и конъюнкция любого числа переменных могут быть реализованы одним нейроном.

Для функций произвольных переменных соответствующие результаты кже не настолько просты, поэтому мы ограничимся констатацией факта, что достаточно сложная сеть с заданной точностью аппроксимирует любую непрерывную функцию.

4.4.3 Обучение сети

На рис. 4.3 для примера приведена схема двухслойной полно связной сети.

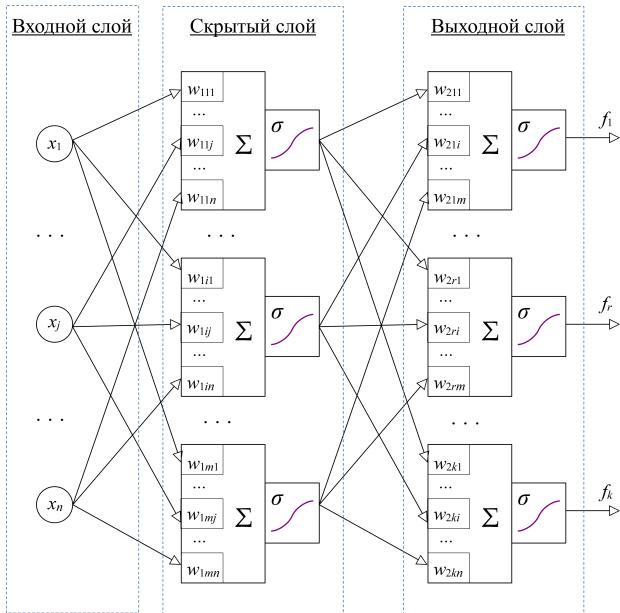


Рис. 4.3: Схема двухслойной полносвязной сети.

Если стоит задача классификации на несколько классов, то для каждого класса на выходном слое помещается нейрон выход которого преобразуется в вероятность принадлежности объекта этому классу.

Для получения вероятности на выходном слое в качестве функции активации нужно использовать softmax. Заметим, что получившаяся архитектура будет немного отличаться от изображённой на схеме, а именно, функция активации на выходном слое будет применяться не независимо к выходу каждого нейрона, а ко всем выходам вместе. Функцию softmax можно представлять и иначе: к выходу каждого нейрона применяется функция активации в виде экспоненты, а затем сумма выходов нормируется на единицу.

Если классов два, то эта схема будет эквивалентна сети с одним выходным нейроном с логистической функцией активации.

Поставим задачу обучения сети.

Обучение подразумевает минимизацию функции потерь на обучающей выборке. Для начала будем минимизировать потери на одном объекте выборки, который обозначим (x, y) . Мы для него не используем индексов, чтобы не загромождать обозначения.

Итак, нужно минимизировать функцию

$$L(f(x), y),$$

где

$$\begin{aligned} f(x) &= (f_1(x), \dots, f_k(x)), & f_r(x) &= \sigma(z_{2r}), \\ z_{2r} &= \sum_{i=1}^m w_{2ri} \sigma(z_{1i}), & z_{1i} &= \sum_{j=1}^n w_{1ij} x_j. \end{aligned}$$

Минимизацию будем вести градиентным методом в пространстве параметров w_{lij} , для чего нам понадобится вычислять производные по этим параметрам.

Вычислим производную по параметрам последнего слоя

$$\begin{aligned} \frac{\partial L(f(x), y)}{\partial w_{2ri}} &= L'_r(f(x), y) \cdot \frac{\partial f_r(x)}{\partial w_{2ri}} = \\ &= L'_r(f(x), y) \sigma'(z_{2r}) \sigma(z_{1i}). \end{aligned}$$

Всё, что нам нужно знать для вычисления данной производной, это z_{lij} — сигналы внутри сети, для получения которых нужно всего лишь подать на вход сети рассматриваемый объект.

Вычислим производную по параметрам скрытого слоя

$$\begin{aligned}\frac{\partial L(f(x), y)}{\partial w_{1ij}} &= \sum_{r=1}^k L'_r(f(x), y) \cdot \frac{\partial f_r(x)}{\partial w_{1ij}} = \\ &\sum_{r=1}^k L'_r(f(x), y) \sigma'(z_{2r}) w_{2ri} \sigma'(z_{1i}) x_j.\end{aligned}$$

Видим (сравнив полученную формулу с предыдущей), что производные выражаются через уже вычисленные производные последнего слоя.

Такой процесс называется процедурой обратного распространения ошибок — когда производные по параметрам некоторого слоя выражаются через производные последующих слоёв. Это позволяет эффективно вычислить градиент по всем параметрам, пройдя сеть в обратном направлении.

4.4.4 Архитектуры нейронных сетей

Сети прямого распространения:

- полносвязные;
- рекуррентные;
- свёрточные.

Сети с памятью.

Мы дали определение нейросети как суперпозиции функций. И все реально используемые сети под такое определение подходят.

Но возможно нарисовать такую сеть нейронов, что она не будет являться суперпозицией функций.

Для этого достаточно соединить нейроны так, чтобы получилась петля, т.е. такой путь по сети, когда сигнал повторно попадает в нейрон, где он уже был.

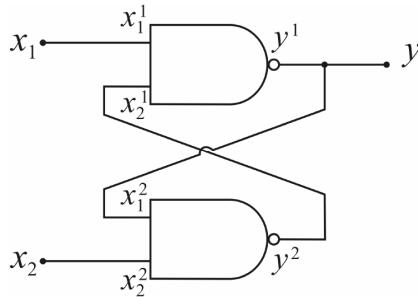


Рис. 4.4: Схема триггера из логических элементов.

Простейшим примером такой сети является всем известный триггер 4.4.

Триггер как нейросеть с памятью

Как известно, триггер — это элементарная ячейка памяти, т.е. то, что позволяет хранить 1 бит информации. И это хранение обеспечивается именно петлями в связях.

Вспомним, что есть сети с архитектурой LSTM, который как раз называются сетями с памятью.

Значит ли это, что эффект памяти можно реализовать без петель? Нет. То, что реализовано в LSTM — это не память, а всего лишь медленно распространяющиеся сигналы. Но поскольку такие сигналы уже общепринято называть памятью, нам придётся этому следовать, но держать в голове, что это «не настоящая» память.

Можно пофантазировать, что петли в живых нейронных сетях как раз и отвечают за кратковременную память.

4.4.5 Глубокое обучение

4.4.6 Бустинг и нейронные сети

В задачах с табличными данными нейронные сети, как правило, уступают (и существенно) бустингу в точности. В справедливости этого наблюдения читатель может убедиться, проанализировав результаты соревнований на kaggle.

Ещё раз подчеркнём, что этот вывод касается только задач с табличными данными. Если данные представляют собой изображения, сложные сигналы, текст, то нейронные сети вне конкуренции.

Автору известны лишь отдельные примеры, когда нейросеть по точности превосходила бустинг.

Один из таких примеров представлял задачу с большим количеством целевых переменных. Не путать с многоклассовой классификацией, когда переменная одна, а значений классом — много. Здесь именно случай нескольких целевых переменных (например, прогнозируем одновременно и образование, и профессию человека).

Проблема с такими задачами в том, что существующие реализации бустинга не способны строить один ансамбль деревьев сразу для всех целевых переменных. Они строят свои деревья для каждой целевой переменной. На самом деле, ситуация ещё хуже, и большинство реализаций даже обычную многоклассовую задачу решает методом «один против всех», и лишь некоторые реализации умеют строить один ансамбль деревьев на все классы.

Заметим, что это не проблема бустинга как подхода, а именно ограничения реализаций метода.

Особенности бустинга, отличающие его от нейросетей.

- Делает отбор переменных.
- Может эксплуатировать независимость.
- Не корректирует уже построенные деревья.

Первые два пункта безусловно являются достоинствами.

С последним — не всё так очевидно. На первый взгляд, кажется, что это искусственное ограничение, принятое для простоты реализации. И если его снять (т.е. при добавлении дерева корректировать и предыдущие), то качество должно возрасти.

Однако, более вероятно, что эта особенность — также работает в пользу эффективности бустинга.

Заметим, что произвольное дерево решений может быть (точно) представлено нейросетью соответствующей архитектуры. До-

статочно вспомнить, что дерево представляется с помощью логических функций и предикатов, а это всё легко представимо нейросетью.

Получается, что дерево решений, а также ансамбль деревьев могут быть заданы в форме нейросети, т.е. являются частными случаями нейросети.

Это так, но из этого не следует, что нейросеть может заменить бустинг. Всё дело в том, что в машинном обучении важен не только (и не столько) класс решающих функций, но и метод обучения. Например, и дискриминант фишера, и метод опорных векторов — оба работают в классе линейных функций, но решения дают разные.

Нейросеть подходящей архитектуры вполне может совпасть как решающая функция с ансамблем деревьев — но в процессе обучения на реальных данных такого совпадения (почти наверное) не произойдёт.

Это примерно так же, как обезьяна, посаженная за печатную машинку, может случайно напечатать «Войну и мир». Но не напечатает.

4.4.7 Свойства нейронных сетей

- Являются универсальным аппроксиматором.
- Решения почти всех методов классификации можно выразить через нейросеть, но нейросеть не заменяет этих методов.
- Имеют меньшую гибкость в способах обучения и в выборе регуляризатора.
- На «табличных» задачах обычно уступают бустингу.
- Во многих других задачах (изображения, текст) не имеют конкурентных альтернатив.
- В задачах с нулевым байесовским уровнем ошибки переобучение менее критично.

4.5 Ансамбли на основе различных методов

Рассмотрим теперь ансамбли, в которых базовые решения строятся разными методами.

В этом случае не возникает проблемы построения разных решений на одной выборке.

Достаточно взять различные доступные алгоритмы. Также можно повторно использовать один и тот же алгоритм, задавая различные параметры. Мы будем формально считать такие варианты разными методами.

4.5.1 Простое голосование

Самой простой способ построение ансамбля — это просто усреднить предсказания базовых методов. Такой подход называется блендинг (*blending*).

Однако и в этом случае требуется аккуратность.

Предположим, что мы решаем задачу классификации с двумя классами и у нас есть три базовых метода. Пусть для какого-то объекта первые два метода оценили вероятность принадлежности объекта классу 1 как 0,2, а третий метод оценил эту вероятность как 0,99.

Если просто взять среднее арифметическое трёх вероятностей, то получим 0,46, т.е. будет приписан класс -1 .

Однако усреднять непосредственно вероятности в данном случае — безусловно плохая идея. При этом не учитывается, что чем ближе вероятность к 0 или 1, тем более значимы различия. Иными словами вероятности 0,99 и 0,999 отличаются сильнее, чем 0,5 и 0,6.

Вспомним, что при рассмотрении наивного байесовского классификатора мы уже сталкивались с подобной задачей. Там мы агрегировали прогнозы по независимым переменным. И тогда мы получили формулу, по которой индивидуальные прогнозы надо было подвергнуть обратному логистическому преобразованию, затем сложить, а в конце применить прямую логистическую функцию.

Если проделать эти действия для вероятностей 0,2, 0,2 и 0,99, то получим 0,86.

Однако эта формула справедлива для независимых переменных, но решения, полученные разными методами нельзя считать независимыми, поскольку они построены по одной и той же выборке (хотя вопрос не так прост).

Разумным решением в такой ситуации кажется использовать логистическое преобразование, но не сладывать преобразованные величины, а усреднять их (что мы, собственно изначально и собирались делать). По такому способу для рассмотренного примера мы получим 0,65. Будет приписан класс 1, но уверенность невысокая (что разумно, поскольку два метода проголосовали «против», и только высокая уверенность третьего метода склонило решение к 1 классу).

Разумеется, веса методов при усреднении не обязательно брать равными. Их можно задавать эвристически (исходя из опыта применения каждого метода).

Также разумный вариант — задавать веса в соответствии с качеством, которое метод достигает на кроссвалидации (о методе кроссвалидации будет соответствующий раздел).

Также напрашивается идея подбирать веса, напрямую минимизируя потери на выборке. Для этого подойдёт уже знакомый метод логистической регрессии. Но это уже тема следующего подраздела.

4.5.2 Стэкинг

Стэкинг (stacking) — довольно популярный метод ансамблирования. А если учесть, что target encoding — это фактически частный случай блендинга — то весьма популярный.

Идея заключается в том, что мы решаем задачу разными методами, как в блендинге, но полученные прогнозы не усредняем, а используем в качестве новых переменных.

Новые переменные мы используем «обычным образом», т.е. на них строим решение, совершенно так же, как на любых других переменных.

Эти новые переменные могут как заменять исходные, так и добавляться к ним.

Важный момент: прогнозы базовыми алгоритмами нужно строить в режиме кроссвалидации. Подробно эта схема описана в соответствующем разделе.

Строго говоря, для target encoding также нужно использовать скорректированные оценки частот, например на основе leave one out (описан с последующих разделах), либо с использованием регуляризации.

Заметим, что для получения новых переменных в стэкинге нам не обязательно (базовыми алгоритмами) прогнозировать целевую переменную. Можно прогнозировать любые переменные. Это довольно экзотический вариант, но иногда подобные приёмы срабатывают.

4.6 Выводы

Практически на всех задачах анализа данных лучшие результаты достигаются именно ансамблевыми методами.

Этот факт кажется вполне естественным.

Например, известно, что если провести несколько измерений какой-либо величины, а затем взять среднее, то ошибка уменьшится.

Это легко доказывается в теории вероятностей: дисперсия среднего независимых одинаково распределённых случайных величин в N раз меньше дисперсии каждой величины.

Но за счёт усреднения уменьшается только случайная ошибка, систематическая ошибка остаётся прежней. Похожие рассуждения используются в разложении на bias и variance (смещение и разброс). В дальнейшем мы подробно рассмотрим это разложение и обсудим, насколько оно объясняет эффективность ансамблей.

В примере с уменьшением ошибки при усреднении результатов измерений есть ключевой момент: ошибки измерений предполагаются независимыми.

В противоположность этому, решения, используемые в ансамбле, являются зависимыми, поскольку строятся на основе одной

и той же выборки. Поэтому пример с независимыми измерениями совершенно не годится для объяснения эффективности ансамблей решающих функций.

Также часто как пример эффективности коллектива приводится эксперимент, который провёл Фрэнсис Гальтон. В этом эксперименте всем желающим предлагалось на глаз оценить вес животного. Оказалось, что отдельные оценки имели значительный разброс, однако среднее значение оказалось очень близко к правильному значению.

Что этот эксперимент показывает? Только то, что мы и так знаем: при усреднении независимых измерений ошибка уменьшается.

Можно ли считать оценки участников эксперимента независимыми? Скорее, да, поскольку друг с другом они не советовались.

Можно ли считать эти оценки измерениями? Давайте проинтерпретируем их как результат работы нейросети. Ведь человеческий мозг — нейросеть, которая обучается в процессе жизни. Получается, что в эксперименте мы имеем ансамбль нейросетей. Но можно ли назвать полученное решение блендингом? Нет. Поскольку каждая нейросеть обучалась на своих уникальных данных (у каждого человека свой уникальный опыт).

Данное рассуждение призвано показать, что бытовая интуиция, выражаемая поговоркой «одна голова хорошо, а две — лучше», на самом деле не имеет отношения к ситуации с ансамблевыми методами.

Если повторением измерений мы можем случайную ошибку сколь угодно приблизить к нулю, то добавлением методов в ансамбль погрешность можно уменьшить только до определённого предела. Всё дело в том, что выборка одна и добавление методов не добавляет новой информации. Всё, на что мы можем, рассчитывать — это более полно использовать имеющуюся информацию.

Если говорить о практическом применении блендинга, то он как правило действительно повышает точность, но зачастую не очень существенно. Для победы в конкурсе даже маленькая прибавка важна, поэтому в конкурсах блендинг и стэкинг применяются очень часто. Однако при решении реальных задач их применение

ние часто оказывается неоправданным, поскольку небольшая прибавка точности не компенсирует радикального повышения сложности и громоздкости решения.

4.6.1 Бустинг и случайный лес

Методы существенно различаются в настройке.

- Для бустинга параметры сложности — глубина дерева и количество деревьев.
- Для random forest сложность не увеличивается с ростом числа деревьев.

Как правило, бустинг использует деревья меньшей глубины.

Как правило, бустинг достигает лучшего качества.

Свойства ансамблей деревьев:

- работа в разнотипном пространстве,
- работа с пропусками,
- интерпретируемость решений теряется.

Бустинг на деревьях — один из лучших методов.

4.7 Примеры кода

4.7.1 Подбор сложности ансамбля

Следующий пример кода демонстрирует построение ансамблей деревьев.

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import (GradientBoostingClassifier,
                               RandomForestClassifier)
from sklearn.model_selection import KFold
```

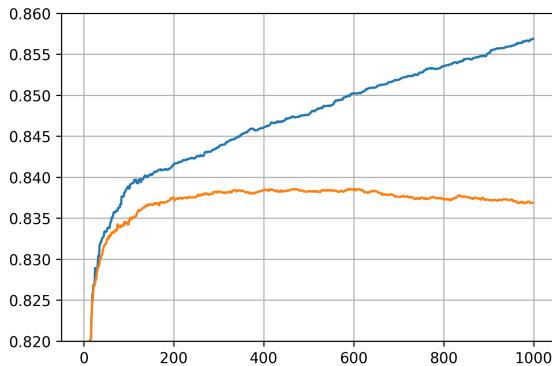


Рис. 4.5: Точность на обучающей выборке и кроссвалидации.

```

data = pd.read_csv('https://archive.ics.uci.edu/ml/' +
                   'machine-learning-databases/adult/adult.data',
                   header=None)
X = data[[2,4,10,11,12]]
Y = (data[14]== '>50K').astype(int)
print(X.shape)

def accuracy(P,Y):
    return (P==Y).sum()/Y.shape[0]

print(accuracy(0,Y))

def crossval(X,Y,model,n_splits):
    kf = KFold(n_splits, shuffle=True, random_state=0)
    scor = []
    for train, test in kf.split(X):
        clf = model.fit(X.iloc[train],Y.iloc[train])
        q = np.array([accuracy(P,Y.iloc[test])])
        for P in clf.staged_predict(X.iloc[test]))
    return np.array(scor).mean(axis=0)

clf = GradientBoostingClassifier(n_estimators=1000,
                                 learning_rate=0.1, max_depth=3, random_state=0)
clf = clf.fit(X,Y)
score_train = np.array([accuracy(P,Y)
                       for P in clf.staged_predict(X)])
score_cv = crossval(X,Y,clf,5)

```

```
plt.figure(figsize=(6,4))
plt.plot(score_train)
plt.plot(score_cv)
plt.ylim((0.82,0.86))
plt.grid()
plt.show()
```

Поясним основные действия.

Используются данные известной задачи Adult из репозитория UCI. Таблица загружается непосредственно с сайта.

Достоинством этой задачи является достаточно большой объём выборки (32561 строка).

Чтобы не загромождать пример кодированием категориальных признаков, выбираем из таблицы только числовые переменные.

Последняя переменная — целевая. Она принимает два значения, которые закодируем как 0 и 1.

Заметим, что классы имеют умеренный дисбаланс, и если всегда приписывать класс 0, то получим точность прогноза 0,76. Это значение выводится на печать и является базовым уровнем точности, с которым будем сравнивать дальнейшие результаты.

В результате запуска фрагмента кода получим диаграмму приведённую на рис. 4.5, где приведены зависимости точности решения на обучающей выборке и кроссвалидации от числа деревьев.

Как и следовало ожидать, точность на обучающей выборке (почти) монотонно растёт, а на кроссвалидации имеет один выраженный максимум.

Можно задать вопрос, как получается, что точность на обучающей выборке ведёт себя не вполне монотонно, хотя метод бустинга в силу самой своей конструкции потери на обучении может только уменьшать. Дело в том, что функция потерь в данном примере не число ошибок, а logloss, и он действительно не возрастает (на обучении) с добавлением деревьев.

Заметим, что для получения данной зависимости не требуется запускать обучение заново для каждого числа деревьев. Достаточно обучить ансамбль один раз при максимальном числе деревьев, а затем воспользоваться функцией `staged_predict`, которая позволяет получать решения на подансамблях.

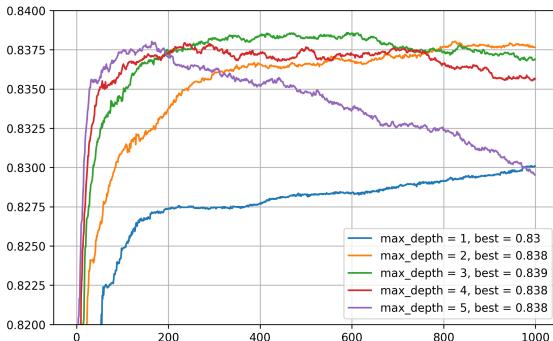


Рис. 4.6: Точность на кроссвалидации при разной глубине деревьев.

Однако, сложность решения методом бустинга зависит не только от числа деревьев, но и от их размера.

Поэтому при решении задач необходимо подбирать как число деревьев, так и глубину (или другой связанный с ней параметр).

Для подбора глубины дерева запустим следующий фрагмент кода (он является продолжением предыдущего).

```

scores = []
for depth in [1,2,3,4,5]:
    print(depth)
    clf = GradientBoostingClassifier(n_estimators=1000,
                                      learning_rate=0.1, max_depth=depth,
                                      random_state=0)
    scores[depth] = crossval(X,Y,clf,5)

plt.figure(figsize=(8,5))

for depth, sc in scores.items():
    plt.plot(sc,label='max_depth = %i, best = %.3g' %
              (depth, sc.max()))
plt.ylim((0.82,0.84))
plt.legend()
plt.grid()

```

Глядя на рис. 4.6, можно сделать вывод, что оптимальным выбором будет глубина 3 при числе деревьев 500.

Однако для глубины 1 мы не достигли максимума точности, и следует увеличить число деревьев (также целесообразно увеличить скорость обучения). Если это проделать, то можно обнаружить, что при скорости обучения 0,2 и 6000 деревьев достигается точность 0,838.

Таким образом, при подборе параметров следует контролировать, что максимум точности «пройден».

Продолжение примера посвящено настройке случайног леса.

```
def crossval_pred(X,Y,model,n_splits):
    kf = KFold(n_splits, shuffle=True, random_state=0)
    q_cv = np.zeros(X.shape[0])
    P = np.zeros(Y.shape)
    for train, test in kf.split(X):
        clf = model.fit(X.iloc[train],Y.iloc[train])
        P[test] = clf.predict(X.iloc[test])
    return P

scores_rf = { 'train': {}, 'cv': {}, 'oob': {}}

for depth in [1,2,3,5,7,10,15,20,30,40,50,60]:
    print(depth)
    model = RandomForestClassifier(n_estimators=200,
                                    max_depth=depth, random_state=0,
                                    oob_score=True).fit(X,Y)
    P = model.predict(X)
    scores_rf['train'][depth] = accuracy(P,Y)
    scores_rf['oob'][depth] = model.oob_score_
    P = model.oob_decision_function_
    P = (P[:,1]>P[:,0]).astype(int)
    scores_rf['oob'][depth] = accuracy(P,Y)
    P = crossval_pred(X,Y,model,5)
    scores_rf['cv'][depth] = accuracy(P,Y)

pd.DataFrame(scores_rf).plot(figsize=(8,5))
plt.grid()
```

Здесь следует обратить внимание на функцию `crossval_pred`, которая делает прогнозы для всех объектах обучающей выборки методом кроссвалидации. Именно про подобный способ прогноза шла речь при описании метода стэкинга.

Для случайног леса увеличение числа деревьев может только улучшать результат, поэтому мы не будем подбирать этот па-

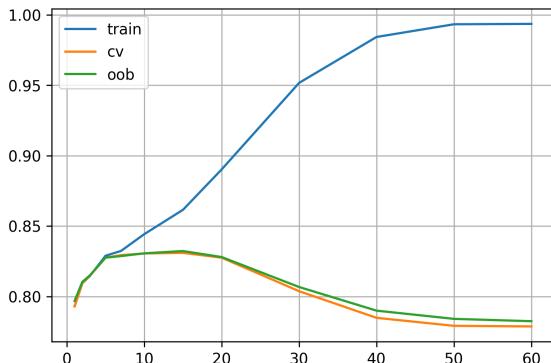


Рис. 4.7: Точность для RandomForest.

метр, а возьмём значение 200, которое обычно бывает «достаточным».

Подбирать будем только глубину деревьев.

На рис. 4.7 показаны зависимости точности от глубины.

Видим, что после глубины 50 точность перестаёт меняться. Это объясняется тем, что фактически более глубокие деревья не строятся, поскольку уже этой глубины достаточно, чтобы классифицировать выборку без ошибок.

Но почему тогда точность на обучении не достигает единицы? Всё дело в том, что каждое дерево строится не по всей выборке и не попавшие в подвыборку объекты могут классифицироваться с ошибкой.

Видим, что оптимальная глубина равна 15. При этом достигаемая точность существенно выше, чем если глубину не ограничивать. Это наблюдение, очевидно, вполне опровергает иногда встречающееся мнение, что метод случайного леса не требует подбора параметров.

Без чего метод RandomForest действительно позволяет обойтись — это без кроссвалидации, поскольку в него встроена возможность получать оценку «out of bag», которая вычисляется на основе объектов которые не были выбраны процедурой bootstrap.

В приведённом коде значение «oob» присваивается дважды.

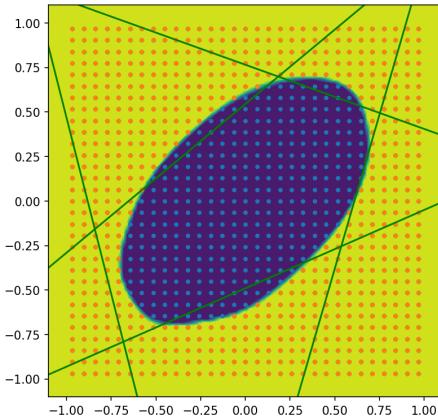


Рис. 4.8: Решение на основе нейросети.

Это не ошибка, поскольку значения в обоих вариантах одинаковы. Первый способ проще, а второй приведён для примера как более универсальный (на его основе можно вычислять другие критерии).

Заметим, что оценка «ооб» отличается от «cv». Это можно объяснить тем, что оценка методом кроссвалидации является смещённой (незначительно). Подробнее это обсуждается в соответствующем разделе.

4.7.2 Визуализация нейронной сети

Попытаемся визуализировать структуру простейшей нейросети.

Для примера используем реализацию нейронных сетей из библиотеки `sklearn`. Выбор обусловлен исключительно тем, что эту библиотеку мы уже использовали, поэтому для запуска кода не потребуется установка новых библиотек. Для решения реальных практических задач нейросетевыми методами целесообразно использовать другие библиотеки.

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.neural_network import MLPClassifier
```

```

from sklearn.inspection import DecisionBoundaryDisplay

def plot_line(ab,c,k=2,ax=None):
    n = np.sqrt((ab*ab).sum())
    a, b = ab/n
    c = c/n
    (plt if ax is None else ax).plot([a*c-b*k,a*c+b*k],
        [b*c+a*k,b*c-a*k],c='green')

M = 31
X = np.array([[i,j] for i in range(M)
              for j in range(M)])
X = ((X+0.5)/M - 0.5) * 2
Y = (X[:,0]*X[:,0] - X[:,0]*X[:,1]
      + X[:,1]*X[:,1] > 0.35).astype(int)

K = 5
clf = MLPClassifier(hidden_layer_sizes=(K,),
                     activation='logistic', solver = 'lbfgs',
                     max_iter=500).fit(X, Y)

_, ax = plt.subplots(figsize=(6,6))
DecisionBoundaryDisplay.from_estimator(clf, X,
                                       response_method='predict_proba',
                                       grid_resolution=200, ax=ax, cmap='viridis')
ax.scatter(X[:,0],X[:,1],s=8)
ax.scatter(X[Y>0,0],X[Y>0,1],s=8)

for i in range(K):
    plot_line(clf.coefs_[0][:,i],clf.intercepts_[0][i],
              ax=ax)

ax.set_xlim((-1.1,1.1))
ax.set_ylim((-1.1,1.1))

```

Код начинается с функции, которая отрисовывает прямую по её каноническому уравнению.

В качестве параметров уравнений прямых берутся коэффициенты скрытого слоя.

На рис. 4.8 мы можем видеть, как нейроны скрытого слоя аппроксимируют прямыми линиями границу между классами. Результирующая граница оказывается слаженной, благодаря свойствам выбранной функции активации.

Для наглядности вместо случайной выборки взята регулярная расстановка точек.

Принадлежность объекта классу определяется в соответствии с принадлежностью точки заданному эллипсу.

Заметим, что получаемое решение сильно варьируется от запуска к запуску, и в части испытаний сеть обучается неудовлетворительно.

Глава 5

Градиентный бустинг

Идея бустинга заключается в том, чтобы каждый последующий базовый классификатор исправлял ошибки предыдущих.

Вторым принципом бустинга является то, что мы не пытаемся улучшить ранее построенные классификаторы, т.е. если базовая решающая функция построена, то она включается в композицию и в дальнейшем не изменяется.

5.1 Алгоритм AdaBoost

Увеличиваем веса объектов, на которых допущена ошибка.

В методе AdaBoost решение строится в виде композиции

$$\lambda(x) = \text{sign}(\beta_T(x)), \quad \beta_T(x) = \sum_{t=1}^T \alpha_t \lambda_t(x).$$

Требуется построить базовые классификаторы $\lambda_t(x)$ и задать их веса α_t так, чтобы минимизировать число ошибок на обучающей выборке, т.е. минимизировать величину $\sum_{i=1}^N I(\lambda(x^i) \neq y^i)$.

Индикаторную функцию потерь можно выразить как

$$I(\lambda(x^i) \neq y^i) = I(\lambda(x^i)y^i \leq 0) = I(z_i \leq 0),$$

где введено обозначение $z_i = \beta_T(x^i)y^i$.

Величина z_i – так называемый отступ (для i -го объекта выборки). Через неё удобно выражать функции потерь.

5.1.1 Непосредственная оптимизация экспоненциальной функции потерь

Попробуем теперь заменить индикаторную функцию потерь на такую функцию, для которой задача минимизации решалась бы аналитически и при этом новая функция аппроксимировала бы исходную.

В качестве такой функции подойдёт e^{-z} .

Заметим, что $I(z \leq 0) \leq e^{-z}$. Обе функции приведены на рис 5.2.

Видим, что минимизируя экспоненциальные потери, мы также минимизируем и индикаторную функцию потерь. Это, конечно, не означает, что минимумы для средних (на выборке) потерь также будут достигаться одновременно. Но это уже привычная ситуация, что вместо исходной функции потерь мы используем другую, которая качественно с ней согласуется (так мы использовали критерий Джини для ветвления дерева).

Итак, теперь мы будем минимизировать величину

$$R_T = \sum_{i=1}^N e^{-z_i} = \sum_{i=1}^N e^{-y^i \beta_T(x^i)} = \\ \sum_{i=1}^N e^{-y^i (\beta_{T-1}(x^i) + \alpha_T \lambda_T(x^i))} = \sum_{i=1}^N w_i^{T-1} e^{-y^i \alpha_T \lambda_T(x^i)},$$

где $w_i^{T-1} = e^{-y^i \beta_{T-1}(x^i)}$.

Напомним, что оптимизацию проводим итеративно, т.е. на каждом шаге оптимизируем только очередной базовый классификатор, зафиксировав все предыдущие.

Рассмотрим итерационный шаг. Предположим, что $T-1$ классификаторов уже построено и на очередном шаге нужно оптимизировать только α_T и λ_T .

Отложим пока вопрос, как построить очередной классификатор λ_T и предположим, что какой-то λ_T у нас есть. Требуется подобрать α_T .

Вспомним, что λ_T принимает только значения $+1$ и -1 . Пере-

группировав слагаемые, получим

$$R_T = W_+^T e^{-\alpha_T} + W_-^T e^{\alpha_T},$$

где

$$W_+^T = \sum_{i:y^i \lambda_T(x^i)=1} w_i^{T-1}, \quad W_-^T = \sum_{i:y^i \lambda_T(x^i)=-1} w_i^{T-1}.$$

Дифференцируем

$$\frac{\partial R_T}{\partial \alpha_T} = -W_+^T e^{-\alpha_T} + W_-^T e^{\alpha_T}.$$

Из условия $\frac{\partial R_T}{\partial \alpha_T} = 0$ находим

$$W_+^T e^{-\alpha_T} = W_-^T e^{\alpha_T}, \quad \alpha_T = \ln \sqrt{\frac{W_+^T}{W_-^T}}.$$

Подставив оптимальное значение α_T , находим минимальные потери

$$R_T^* = W_+^T \sqrt{\frac{W_-^T}{W_+^T}} + W_-^T \sqrt{\frac{W_+^T}{W_-^T}} = 2 \sqrt{W_+^T W_-^T}.$$

Заметим, что потери на предыдущем шаге выражаются как

$$R_{T-1}^* = \sum_{i=1}^N w_i^{T-1} = W_+^T + W_-^T.$$

Найдём, насколько потери уменьшились

$$\begin{aligned} R_{T-1}^* - R_T^* &= W_+^T + W_-^T - 2 \sqrt{W_+^T W_-^T} = \\ &= \left(\sqrt{W_+^T} - \sqrt{W_-^T} \right)^2. \end{aligned}$$

Видим, что чем больше различие W_+^T и W_-^T , тем сильнее уменьшаются потери. При этом не важно, какая из величин больше, но если больше W_-^T , то α_T получается отрицательным (фактически это означает, что мы инвертируем прогноз текущего классификатора).

Разберёмся, наконец, как же строить базовые классификаторы λ_T .

Если величины интерпретировать w_i^{T-1} как веса объектов выборки на шаге T , то W_+^T будет иметь смысл суммарного веса правильно классифицированных классификатором λ_T объектов, а W_-^T – суммарный вес неправильно классифицированных объектов.

И чтобы эти величины максимально различались, достаточно строить λ_T как решение обычной задачи классификации с весами.

Таким образом, AdaBoost позволяет в качестве базового метода классификации использовать абсолютно любую готовую реализацию любого метода, которая позволяет задавать веса объектов.

Если на каком-то шаге базовый алгоритм классифицирует выборку без ошибок, то α_T становится бесконечным и AdaBoost не может продолжить работу. Но бустинг и не имеет смысла, если базовый алгоритм не делает ошибок, поскольку задача и так решена.

5.1.2 AdaBoost как градиентный бустинг

Как и ранее, мы будем минимизировать величину

$$R_T = \sum_{i=1}^N e^{-y^i \beta_T(x^i)},$$

но теперь используем представление

$$\beta_T(x^i) = \beta_{T-1}(x^i) + \xi_i,$$

где ξ_i будем считать свободной переменной, по которой и будем вести минимизацию.

На самом деле, ξ_i – это прогноз базового классификатора на объекте x^i , и он не может быть произвольным. Но пока будем считать, что может.

Заметим, что в этом случае минимум даже не существует. Однако, это не помешает нам двигаться градиентным методом в сторону уменьшения потерь.

Движение против градиента означает, что

$$\xi_i = -\alpha_T \frac{\partial R_T}{\partial \xi_i},$$

где α_T – величина шага. Если раньше мы оптимизировали его впервую очередь, то теперь сделаем это потом.

Теперь нам нужно добиться, чтобы базовый метод давал решение λ_T , приближающее антиградиент. Это можно сделать, если на базовом уровне решать задачу регрессии.

Но чтобы получить именно AdaBoost, на базовом уровне мы должны поставить задачу классификации, когда λ_T принимает значения $+1, -1$.

Решение λ_T можно интерпретировать как вектор, задающий направление, вдоль которого будет смещаться композиция. А чтобы уменьшалась функция потерь, это смещение должно быть примерно вдоль антиградиента.

Скорость уменьшения потерь определяется величиной скалярного произведения направления сдвига и антиградиента, а именно

$$-\sum_{i=1}^N \frac{\partial R_T}{\partial \xi_i} \lambda_T(x^i) \rightarrow \max.$$

Вычислим градиент для экспоненциальных потерь

$$\frac{\partial R_T}{\partial \xi_i} = -y^i e^{-y^i \beta_{T-1}(x^i)} = -y^i w_i^{T-1}$$

и подставим в скалярное произведение. Получим задачу максимизации

$$\sum_{i=1}^N y^i w_i^{T-1} \lambda_T(x^i) \rightarrow \max.$$

Видим, что это ровно та же задача классификации, которая решается базовым методом в алгоритме AdaBoost.

Этим мы показали, что AdaBoost является частным случаем градиентного бустинга, если использовать экспоненциальную функцию потерь и если сводить поиск антиградиента к задаче классификации.

Последнее является очень важным уточнением. Так например, если взять реализацию градиентного бустинга из scikit-learn, и выбрать в ней экспоненциальные потери, то в результате AdaBoost мы не получим, поскольку градиентный бустинг из scikit-learn на базовом уровне использует деревья регрессии, а не классификации.

Кастомизацией функции потерь в XGBoost также нельзя получить алгоритм, совпадающий с AdaBoost.

5.1.3 Сходимость процесса бустинга

Если бустинг не останавливать, то он будет стремиться оценить функцию условной вероятности.

- Если в точке пространства находится один объект выборки, то эмпирическая условная вероятность соответствующего класса в этой точке равна 1.
- Бустинг приближает вероятность через логистическую функцию.

Если условные вероятности нигде не равны 0 или 1, то бустинг сходится (веса деревьев стремятся к 0).

Полезно исследовать поведение методов не только на выборке, но и на распределениях.

Оценивание условной вероятности Условную вероятность $g(x) = P(y = 1 | x)$ представим как находящиеся в точке x два объекта: класса 1 с весом $w_0 g(x)$ и класса -1 с весом $w_0(1 - g(x))$.

В результате выполнения бустинга вес первого объекта станет равным

$$w^{+1}(x) = w_0 g(x) \cdot A e^{-\beta(x)},$$

где константа A есть произведение всех нормировочных множителей.

Конечный вес второго объекта есть

$$w^{-1}(x) = w_0(1 - g(x)) \cdot A e^{\beta(x)}.$$

Если приравнять веса объектов, то получим

$$g(x) = \frac{1}{1 + e^{-2\beta(x)}}.$$

5.2 Градиентный бустинг

Фактически мы уже познакомились с градиентным бустингом, когда использовали его идею для вывода алгоритма AdaBoost. Рассмотрим теперь его в более общем виде.

Итак, у нас есть некоторая функция потерь $L(\beta(x), y)$ и мы хотим минимизировать средние потери на выборке

$$R(S, \beta_T) = \sum_{i=1}^N L(\beta_T(x^i), y^i).$$

Пусть мы уже построили $T - 1$ базовых решающих функций и требуется построить очередную функцию $\lambda_T(x)$. Обозначим $\xi_i = \lambda_T(x^i)$ и будем считать ξ_i свободными переменными.

Попробуем провести минимизацию потерь методом градиентного спуска (в пространстве переменных ξ_i). Вычислим производную

$$\frac{\partial R(S, \beta_T)}{\partial \xi_i} = \frac{\partial L(\beta_T(x^i), y^i)}{\partial \xi_i} = \frac{\partial L(\beta_{T-1}(x^i) + \xi_i, y^i)}{\partial \xi_i}.$$

Обозначим через ω^T значение антиградиента в точке $\xi = 0$. Имеем для i -й компоненты вектора ω^T

$$\omega_i^T = - \left. \frac{\partial R(S, \beta_T)}{\partial \xi_i} \right|_{\xi_i=0} = - \left. \frac{\partial L(u, y^i)}{\partial u} \right|_{u=\beta_{T-1}(x^i)}.$$

Компоненты антиградиента известны только для точек выборки. Для обобщения на всё пространство требуется применить базовый метод классификации (для краткости будем говорить о классификации, однако все рассуждения справедливы для общей задачи построения решающей функции).

Итак, очередное базовое решение $\lambda_T(x)$ в точках выборки должно близким к ω^T .

Значения $\lambda_T(x)$ на выборке удобно представлять в виде вектора. Обозначим через $\tilde{\lambda}_T$ вектор с компонентами $\lambda_T(x^i)$.

Получается, что критерием качества для базового метода должна быть близость векторов $\tilde{\lambda}_T$ и ω^T .

Очевидным кажется определить близость векторов через евклидову норму их разности. В этом случае базовый метод должен решать задачу регрессии с функцией потерь в виде квадрата отклонения.

Однако вспомним, что бы хотим реализовать градиентный спуск. В этом случае нам важно, насколько точно мы можем выдерживать направление градиента. А точность направления, очевидно, определяется углом между вектором градиента и вектором, в направлении которого мы фактически движемся.

Минимизация угла между векторами эквивалентна максимизации скалярного произведения векторов при фиксированной норме. Таким образом, базовый метод может решать задачу регрессии с критерием

$$\begin{cases} \tilde{\lambda}_T \omega^T \rightarrow \max \\ (\tilde{\lambda}_T)^2 = 1 \end{cases}.$$

Может показаться, что задачи минимизации квадрата разности и максимизации скалярного произведения эквивалентны. При фиксированной норме искомого вектора это действительно так, поскольку

$$(\tilde{\lambda}_T - \omega^T)^2 = (\tilde{\lambda}_T)^2 + (\omega^T)^2 - 2\tilde{\lambda}_T \omega^T.$$

Если же норму не фиксировать, то результаты могут существенно различаться.

На рис. 5.1 приведён условный пример, где a выступает в роли вектора градиента, к которому мы ходим приблизиться, а F – множество решений, которые способен построить базовый алгоритм. Видим, что с точки зрения минимума квадрата отклонения ближайшим к a из допустимых решений будет вектор b , а минимальный угол с вектором градиента обеспечивает вектор c .

Заметим, что базовый метод может решать задачу классификации, даже если сам бустинг решает задачу регрессии.

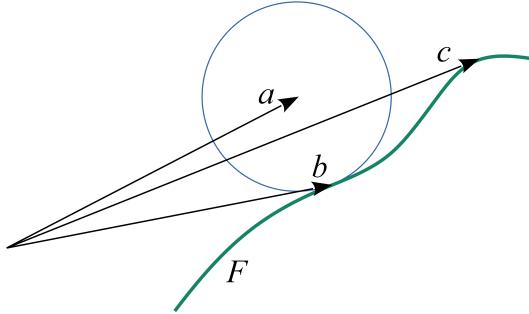


Рис. 5.1: Различные приближения к вектору градиента.

Действительно, задача

$$\begin{cases} \tilde{\lambda}_T |\omega^T| \operatorname{sign}(\omega^T) \rightarrow \max \\ (\tilde{\lambda}_T)_i \in \{-1, 1\} \end{cases}$$

является задачей классификации с целевой функцией $\operatorname{sign}(\omega^T)$ и весами $|\omega^T|$.

И поставить задачу в такой форме мы можем для любой исходной задачи.

Следует отметить важный момент, на который редко акцентируют внимание.

В методах бустинга мы имеем в общем случае четыре различных функции потерь:

- исходная функция потерь, которую мы должны минимизировать (например, число ошибок классификации),
- функция потерь, которую мы решили минимизировать вместо исходной (например, экспоненциальная функция потерь для AdaBoost),
- функция потерь, которую должен минимизировать базовый алгоритм (например, для AdaBoost это ошибка классификации с учётом весов объектов),

- функция потерь, которую мы будем минимизировать базовым алгоритмом вместо той, что получилась в предыдущем пункте (например, для AdaBoost на деревьях решений разумным выбором будет Gini impurity или logloss).

5.3 Метод XGBoost

Данный метод использует всего одну функцию потерь.

Как и ранее, задана функция потерь $L(\beta(x), y)$ и мы хотим минимизировать средние потери на выборке

$$R(S, \beta_T) = \sum_{i=1}^N L(\beta_T(x^i), y^i).$$

Решение на текущем шаге выражается через решение на предыдущем шаге как

$$\beta_T(x^i) = \beta_{T-1}(x^i) + \xi_i.$$

Найдём первую и вторую производные функции потерь в точке $\xi = 0$

$$g_i = \frac{\partial R(S, \beta_T)}{\partial \xi_i} \Big|_{\xi_i=0} = \frac{\partial L(u, y^i)}{\partial u} \Big|_{u=\beta_{T-1}(x^i)}.$$

$$h_i = \frac{\partial^2 R(S, \beta_T)}{\partial (\xi_i)^2} \Big|_{\xi_i=0} = \frac{\partial^2 L(u, y^i)}{\partial u^2} \Big|_{u=\beta_{T-1}(x^i)}.$$

Разложим функцию потерь в ряд Тейлора до второго порядка

$$L(\beta_T(x^i), y^i) = L(\beta_{T-1}(x^i), y^i) + g_i \xi_i + \frac{1}{2} h_i \cdot (\xi_i)^2.$$

Метод XGBoost в качестве базового метода использует деревья решений.

Для построения дерева нужно иметь критерий ветвления, для чего требуется уметь оценивать качество любых вариантов дерева. А поскольку критерий для дерева — это сумма критериев по вершинам, достаточно понять, как оценить решение в произвольной вершине дерева.

Дерево решений в каждой вершине приписывает одинаковые значения для всех объектов, попавших в эту вершину. Обозначим этот прогноз через ζ .

Запишем потери в вершине через разложение в ряд

$$R(J, \zeta) = \sum_{i \in J} L(\beta_{T-1}(x^i, y^i) + g_i \zeta + \frac{1}{2} h_i \zeta^2,$$

где J – множество номеров объектов, попавших в вершину.

Преобразуем к виду

$$R(J, \zeta) = G(J) \cdot \zeta + \frac{1}{2} H(J) \cdot \zeta^2 + C(J),$$

где

$$\begin{aligned} G(J) &= \sum_{i \in J} g_i, & H(J) &= \sum_{i \in J} h_i, \\ C(J) &= \sum_{i \in J} L(\beta_{T-1}(x^i, y^i)). \end{aligned}$$

Из условия $\frac{\partial R(J, \zeta)}{\partial \zeta} = 0$ находим оптимальный прогноз в вершине

$$\zeta^* = -\frac{G(J)}{H(J)}.$$

Потери в вершине при оптимальном прогнозе составят

$$\begin{aligned} R^*(J) &= R(J, \zeta^*) = \\ &= -G(J) \cdot \frac{G(J)}{H(J)} + \frac{1}{2} H(J) \cdot \left(-\frac{G(J)}{H(J)} \right)^2 + C(J) = \\ &= -\frac{1}{2} \cdot \frac{G^2(J)}{H(J)} + C(J). \end{aligned}$$

Заметим, что сумма $C(J)$ по всем вершинам не зависит от структуры дерева (поскольку вершины в сумме покрывают всю выборку), поэтому критерий ветвления от $C(J)$ не зависит.

Мы получили, что критерий ветвления при построении дерева должен максимизировать сумму по всем вершинам величин

$$\frac{G^2(J)}{H(J)}.$$

В критерий ветвления в методе XGBoost также входит слагаемое, отвечающее за регуляризацию. Чтобы не перегружать выкладки, мы его не использовали.

Найдём теперь конкретные выражения полученного критерия ветвления для некоторых функций потерь. Начнём с logloss, которая представляет собой функцию правдоподобия и требует решения в виде оценки вероятности класса.

Используя композицию деревьев, мы можем оценить условную вероятность в форме логистической функции от суммы прогнозов, т.е.

$$g(x) = \frac{1}{1 + e^{-\beta(x)}}.$$

Выразим теперь логарифмическую функцию потерь

$$\begin{aligned} L(y, g(x)) &= \begin{cases} -\ln g(x), & y = 1 \\ -\ln(1 - g(x)), & y = -1 \end{cases} \\ &= \ln \left(1 + e^{-y\beta(x)} \right) = -\ln \sigma(y\beta(x)). \end{aligned}$$

Вычислим производные от функции потерь

$$\begin{aligned} \frac{\partial -\ln \sigma(y\beta(x))}{\partial \xi} &= \frac{\partial \ln(1 + e^{-y\beta(x)})}{\partial \xi} = \\ &- \frac{e^{-y\beta(x)}}{1 + e^{-y\beta(x)}} y = -y\sigma(-y\beta(x)), \end{aligned}$$

$$\begin{aligned} \frac{\partial^2 -\ln \sigma(y\beta(x))}{(\partial \xi)^2} &= \frac{\partial -y\sigma(-y\beta(x))}{\partial \xi} = \\ &y^2 \frac{e^{y\beta(x)}}{(1 + e^{y\beta(x)})^2} = y^2 \sigma(y\beta(x))\sigma(-y\beta(x)) = \sigma(\beta(x))\sigma(-\beta(x)). \end{aligned}$$

Мы учли, что $y^2 = 1$.

Получаем

$$g_i = \frac{\partial L(u, y^i)}{\partial u} \Big|_{u=\beta_{T-1}(x^i)} = -y^i \sigma(-y^i \beta_{T-1}(x^i)).$$

$$h_i = \frac{\partial^2 L(u, y^i)}{(\partial u)^2} \Big|_{u=\beta_{T-1}(x^i)} = \sigma(\beta_{T-1}(x^i))\sigma(-\beta_{T-1}(x^i)).$$

Для экспоненциальных потерь

$$g_i = \frac{\partial e^{-y^i u}}{\partial u} \Big|_{u=\beta_{T-1}(x^i)} = -y^i e^{-y^i \beta_{T-1}(x^i)} = -y^i w_i.$$

$$h_i = \frac{\partial^2 e^{-y^i u}}{(\partial u)^2} \Big|_{u=\beta_{T-1}(x^i)} = e^{-y^i \beta_{T-1}(x^i)} = w_i.$$

Величину w_i будем считать весом i -го объекта выборки (на текущем шаге). Для отдельной вершины дерева (как и ранее J – множество индексов объектов в этой вершине) просуммируем веса объектов отдельно по классам

$$W_{-1}(J) = \sum_{i \in J} w_i I(y^i = -1) \quad W_1(J) = \sum_{i \in J} w_i I(y^i = 1).$$

Для критерия ветвления

$$\begin{aligned} \frac{G^2(J)}{H(J)} &= \frac{(W_{-1}(J) - W_1(J))^2}{W_{-1}(J) + W_1(J)} = \frac{(W(J) - 2W_1(J))^2}{W(J)} = \\ &= \frac{W(J) - 4W_1(J) + 4\frac{(W_1(J))^2}{W(J)}}, \end{aligned}$$

где $W(J) = W_{-1}(J) + W_1(J)$.

Если просуммировать $W(J)$ или $W_1(J)$ по всем вершинам дерева, то результат не будет зависеть от собственно дерева, поэтому эти слагаемые не влияют на выбор ветвления.

Таким образом, критерий ветвления будет основан на сумме величин

$$\frac{(W_1(J))^2}{W(J)}.$$

Полученный критерий ветвления оказывается полностью эквивалентным критерию Gini impurity. Действительно, для Gini

impurity качество дерева определяется как сумма величин

$$\begin{aligned} W_{-1}(J) \cdot \frac{W_{-1}(J)}{W(J)} + W_1(J) \cdot \frac{W_1(J)}{W(J)} &= \\ \frac{(W(J) - W_1(J))^2}{W(J)} + \frac{(W_1(J))^2}{W(J)} &= \\ W(J) - 2W_1(J) + 2 \frac{(W_1(J))^2}{W(J)}. \end{aligned}$$

Получается, что метод XGBoost при экспоненциальной функции потерь будет на каждом шаге строить дерево по критерию Gini impurity.

Но ведь данный критерий используется для построения деревьев классификации, а в методе XGBoost всегда строится дерево регрессии. Но противоречия здесь нет.

Дело в том, что критерий ветвления — это функция от объектов выборки, попавших в вершину, и его можно уже не рассматривать как функцию от оптимального прогноза, поскольку последний — функция выборки.

Так что мы действительно имеем ситуацию, что дерево регрессии строится по критерию Gini impurity.

Но мы можем представить дерево регрессии из XGBoost, как дерево классификации, в каждой вершине которого бинарный прогноз умножается на некоторый коэффициент $\alpha(J)$.

Получается, что XGBoost при экспоненциальной функции потерь очень похож на AdaBoost, но имеет принципиальное отличие в том, что каждое дерево входит в композицию не просто с весом α_t , а с весом, который для каждой вершины дерева свой. При этом всё дерево дополнительно умножается на некоторый коэффициент, который называется скоростью обучения.

Заметим, что и в AdaBoost ничто не запрещает ввести параметр скорости обучения.

Также в AdaBoost мы можем использовать не только Gini impurity.

Таким образом, XGBoost при экспоненциальной функции потерь вовсе не эквивалентен методу AdaBoost на деревьях, но имеет много общего.

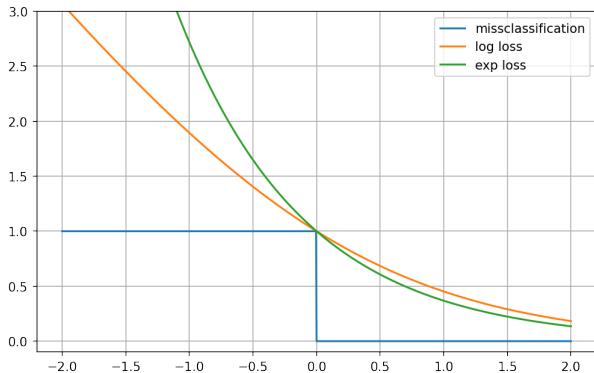


Рис. 5.2: Функции потерь для бустинга.

Заметим, что библиотека XGBoost является очень гибкой, в частности позволяет реализовать также метод случайного леса.

5.4 Свойства бустинга

Градиентный бустинг строит композицию $\beta(x)$, минимизируя функцию потерь на выборке.

Функции потерь

Обобщённый наивный байесовский классификатор Ранее мы для наивного байесовского классификатора получили выражение в виде логистической регрессии

$$g(x) = \sigma \left(u_0 + \sum_{j=1}^n u_j \sigma^{-1}(g_j(x_j)) \right),$$

при $u_0 = (n-1)(\ln p - \ln(1-p))$, $u_j = 1$.

Обобщим это выражение, считая веса свободными параметрами и допуская произвольные оценочные функции. Получим

$$g(x) = \sigma \left(u_0 + \sum_{j=1}^n u_j s(x_j) \right).$$

Бустинг на пороговых классификаторах Бустинг на пороговых классификаторах («пнях») является разновидностью обобщённого наивного байесовского классификатора.

Действительно, каждая $\lambda_t(x)$ в композиции

$$\beta(x) = \sum_{t=1}^T \alpha_t \lambda_t(x))$$

зависит только от одной переменной X_{i_t} , поэтому после группировки слагаемых выражение можно привести к виду

$$2\beta(x) = \sum_{i=1}^n u_i s(x).$$

Подставив в выражение для $g(x)$, получим искомый вид.

Бустинг на деревьях и ряд Бахадура Модель можно естественным образом обобщить по аналогии с рядом Бахадура, включив возможность учитывать зависимости между переменными, последовательно добавляя парные зависимости, зависимости в тройках и т.д.

$$g(x) = \sigma \left(u_0 + \sum_{j=1}^n u_j s_j(x_j) + \sum_{j,k} u_{jk} s_{jk}(x_j, x_k) + \right. \\ \left. + \sum_{j,k,l} u_{jkl} s_{jkl}(x_j, x_k, x_l) + \dots \right).$$

Аппроксимация распределения выборкой

Условная вероятность «чужого» класса в каждой точке равна 0,1.

Решение градиентным бустингом

Глубина дерева 2, решения на основе 15 и 1000 деревьев.

Бустинг на деревьях минимальной глубины

Глубина дерева 1, решения на основе 15 и 1000 деревьев.

Модельный пример

Функция условной вероятности.

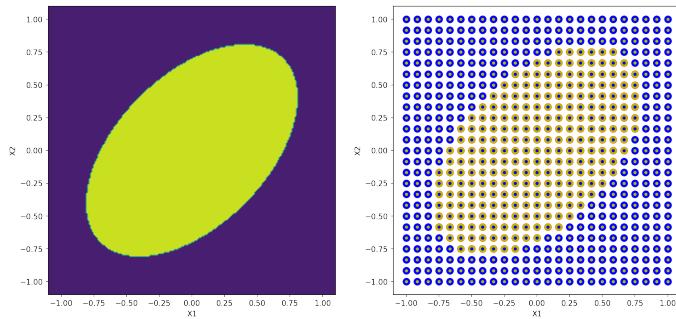


Рис. 5.3: Дерево решений для задачи Iris.

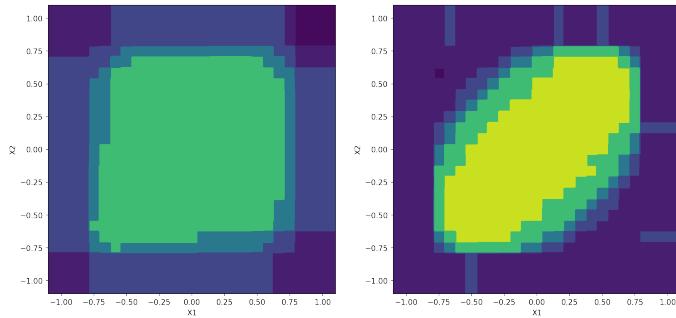


Рис. 5.4: Дерево решений для задачи Iris.

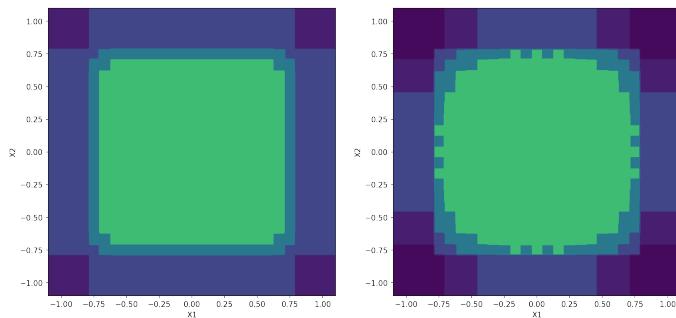


Рис. 5.5: Дерево решений для задачи Iris.

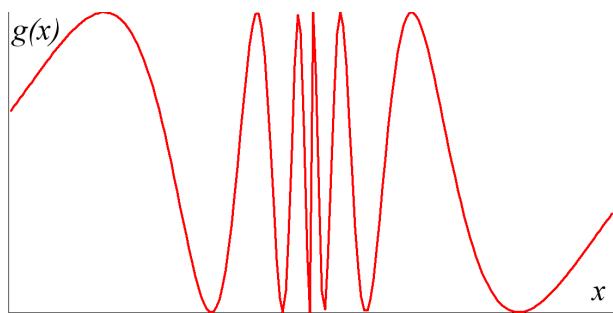


Рис. 5.6: Дерево решений для задачи Iris.

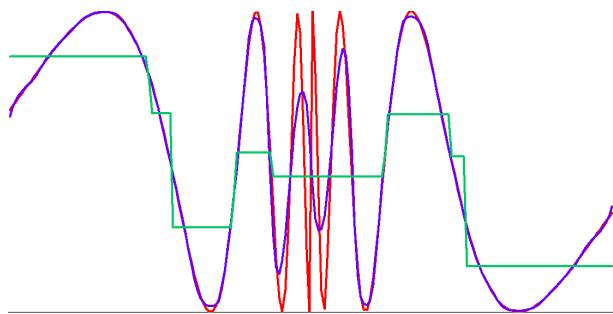


Рис. 5.7: Дерево решений для задачи Iris.

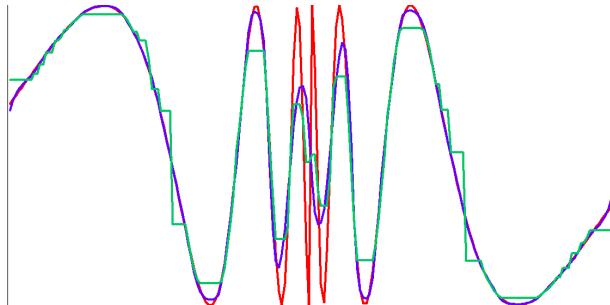


Рис. 5.8: Дерево решений для задачи Iris.

Аппроксимация сплайном

Кубический сплайн на 20 интервалов.

AdaBoost 10 итераций.

Boosting

AdaBoost 100 итераций.

Понятие отступа Иногда для обоснования бустинга вводится понятие отступа.

Отступ есть

$$\theta = \frac{y\beta(x)}{\varkappa}, \quad \varkappa = \sum_{t=1}^T \alpha_t.$$

Из-за нормировки в виде \varkappa сложность композиции влияет на оценку риска.

5.5 Выводы

Проклятие размерности

Для случая независимых переменных «проклятие» размерности превращается в преимущество:

- чем больше зависимых переменных — тем больше требуемый объём выборки,
- чем больше независимых переменных — тем меньше требуемая выборка.

С увеличением числа независимых переменных качество решения только растёт.

Бустинг и случайный лес

Методы существенно различаются в настройке.

- Для бустинга параметры сложности — глубина дерева и количество деревьев.
- Для random forest сложность не увеличивается с ростом числа деревьев.

Как правило, бустинг использует деревья меньшей глубины.

Как правило, бустинг достигает лучшего качества.

Выводы

- Важнейшей причиной эффективности бустинга является использование эффекта независимости (переменных, подпространств, моделей).
- Бустинг на пороговых классификаторах является разновидностью непараметрической логистической регрессии, также его можно считать разновидностью (существенно обобщённого) наивного байесовского классификатора.
- Бустинг реализует «удачный» вариант непараметрической аппроксимации условной вероятности.

Свойства:

- работа в разнотипном пространстве,
- работа с пропусками,
- интерпретируемость решений.

Бустинг на деревьях — один из лучших методов.

5.6 Примеры кода

Приведём пример кода для построения дерева и его визуализации.

В примере также показана возможность сохранения рисунка в файл.

```
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
from sklearn import tree

clf = tree.DecisionTreeClassifier(max_depth=3,
                                  random_state=0)
iris = load_iris()
clf = clf.fit(iris.data, iris.target)

plt.figure(figsize=(8,8))
tree.plot_tree(clf)
plt.savefig('tree_iris.png', dpi=300,
            bbox_inches='tight')
plt.show()
```

Результат выполнения кода приведён на рис. 3.5. Размер (глубина) дерева принудительно ограничен с помощью соответствующего параметра.

Глава 6

Оценивание качества решений

Точечные оценки риска

- эмпирический риск,
- контрольная выборка,
- скользящий экзамен,
- bootstrap,
- out of bag!,
- комбинации (несколько статистик),
- другие статистики.

6.1 Постановка задачи

Пусть X – пространство значений переменных, используемых для прогноза, а Y – пространство значений прогнозируемых переменных.

Предположим, что все переменные являются случайными величинами с некоторой неизвестной функцией совместного распределения $F_c(x, y)$.

В самом общем виде это можно формализовать, введя C – множество всех вероятностных мер на заданной σ -алгебре подмножеств множества $D = X \times Y$.

При каждом $c \in C$ имеем вероятностное пространство: $\langle D, B, \mathsf{P}_c \rangle$, где B – σ -алгебра, P_c – вероятностная мера.

Индекс c здесь – это формально введённый параметр. Дело в том, что все функции распределения привычно обозначать одной и той же буквой F , а вероятности – буквой P , и чтобы различать распределения требуется индекс.

Параметр c будем называть *стратегией природы*.

Решающей функцией (алгоритмом классификации) называется соответствие $\lambda: X \rightarrow Y$, причём λ должна быть измеримой функцией.

Качество принятого решения оценивается заданной функцией потерь $\mathcal{L}: Y^2 \rightarrow [0, \infty)$.

Под риском будем понимать средние потери:

$$R(c, \lambda) = \mathsf{E}\mathcal{L}(y, \lambda(x)) = \int_D \mathcal{L}(y, \lambda(x)) \mathsf{P}_c(dx, dy),$$

$x \in X$, $y \in Y$. Заметим, что значение риска зависит от стратегии природы – распределения, которое неизвестно.

Если использованная нотация в виде интегралов по мере не привычна, можно использовать форму записи

$$\mathsf{E}\mathcal{L}(y, \lambda(x)) = \int_D \mathcal{L}(y, \lambda(x)) dF_c(x, y).$$

Обе формы записи являются универсальными, т.е. позволяют не различать случаи дискретных, (абсолютно) непрерывных и даже сингулярных случайных величин.

Если вторая запись – также непривычна (непонятна) читатель может использовать вместо неё любую форму определения математического ожидания, которая ему знакома.

Цель данной главы – научится оценивать качество решающих функций, что означает построение оценок риска.

В математической статистике все оценки – это (измеримые) функции выборки (такие функции называются статистиками).

В задачах машинного обучения по крайней мере одна выборки у нас всегда есть — это обучающая выборка.

Пусть

$$S = ((x^i, y^i) \in D \mid i = 1, \dots, N)$$

— случайная независимая выборка из распределения P_c , $S \in D^N$.

Тогда любую (измеримую) функцию $\bar{R}(S, \lambda)$ можно считать оценкой риска. И основной задачей главы будет понять, что есть «хорошая» оценка риска и как её построить.

В отличие от классической задачи оценивания параметров оценивание риска подразумевает оценивание случайной величины. Действительно, риск зависит от решающей функции, которая строится по случайно выборке. Значит риск — случайная величина. Формализуем это.

Методом построения решающих функций назовём отображение

$$Q: D^N \rightarrow \Lambda,$$

где Λ — заданный класс решающих функций.

Тогда риск будет величиной $R(c, \lambda_{Q,S})$, где $\lambda_{Q,S}$ — функция, построенная по выборке S методом Q .

Поскольку многие читатели, вероятно, знакомы с работой *The Elements of Statistical Learning* (Hastie T., Tibshirani R., Friedman J.), сделаем некоторые комментарии.

Приведём цитату из упомянутой работы. «Estimation of Err_T will be our goal, although we will see that Err is more amenable to statistical analysis, and most methods effectively estimate the expected error. It does not seem possible to estimate conditional error effectively, given only the information in the same training set.»

Переведём с использованием «наших» обозначений. «Наша цель — оценить риск $R(c, \lambda)$, хотя в статистическом анализе чаще упоминается средний риск $ER(c, \lambda)$, и большинство методов эффективно оценивают именно средний риск. Кажется, что только на основе обучающей выборки невозможно эффективно оценивать риск.»

В данной главе мы вообще не будем касаться среднего риска, поскольку он нас пока не интересует (средний риск будет обсуждаться в главе 2).

даться только в следующей главе). Здесь же мы будем делать именно то, что по утверждению из цитаты невозможно делать эффективно. На самом деле, у нас просто нет выбора, поскольку в практических задачах требуется именно оценка риска, а не среднего риска. Что касается эффективности, то этот вопрос будет обсуждаться.

На самом деле, мы вправе любую статистику формально объявить оценкой чего угодно. Вопрос только в том, насколько «хорошей» будет такая оценка. Вспомним, какие критерии качества оценок существуют.

Как известно, желательные свойства точечных оценок это:

- несмешённость,
- состоятельность,
- эффективность.

Перенёсем эти понятия на случай оценивания риска, с учётом того, что риск — случайная величина.

Оценка $\bar{R}_Q(S) = \bar{R}(S, \lambda_{Q,S})$ называется несмешённой оценкой риска $R_Q(c, S) = R(c, \lambda_{Q,S})$, если

$$\mathbb{E}_S \bar{R}_Q(S) = \mathbb{E}_S R_Q(c, S).$$

Математическое ожидание берётся по всем выборкам.

Оценка $\bar{R}_Q(S)$ является состоятельной, если

$$\bar{R}_Q(S) \xrightarrow[N \rightarrow \infty]{p} R_Q(c, S).$$

Эффективной оценкой в статистике называется несмешённая оценка с минимальной дисперсией.

В нашем случае (когда оценивается случайная величина) такое определение не подойдёт.

Будем исходить из того, что эффективная оценка должна обеспечивать минимальную погрешность, в роли которой будет выступать средний квадрат отклонения. Справедливо следующее разложение

$$\mathbb{E}(\bar{R} - R)^2 = D\bar{R} + DR + (\mathbb{E}\bar{R} - \mathbb{E}R)^2 - 2 \operatorname{cov}(\bar{R}, R).$$

Видим, что помимо дисперии оценки и её смещения на погрешность влияет и её корреляция с оцениваемой величиной.

Эта корреляция часто (в экспериментах) оказывается практически нулевой, однако в ряде случаев (при использовании logloss) наблюдались даже отрицательные значения. Может показаться странным, что измерение некоррелирует с измеряемой величиной (а тем более, имеет отрицательную корреляцию).

Для пояснения рассмотрим наглядный пример. Представим, что мы проводим измерения длины рельса, причём измерения проводим многократно и в разные дни. Из-за температурного расширения и непостоянства температуры сама измеряемая величина (длина рельса) становится случайной.

Если измерение проводим рулеткой, то погрешность измерения имеет систематическую составляющую (например из-за неточности шкалы рулетки) и случайную компоненту, обусловленную неточностью прикладывания, погрешностью при считывании меток, а также температурным удлинением самой рулетки. Последнее слагаемое даст положительную корреляцию с измеряемой величиной. Однако эта корреляция будет незначительной. Также можно представить, что рулетка сделана из материала с отрицательным коэффициентом температурного расширения, и тогда корреляция также будет отрицательной.

Пример весьма условный, но позволяет наглядно понять смысл всех компонент разложения ошибки.

Рассмотрим в качестве точечных оценок риска известные выборочные функционалы качества (их можно называть функционалами, поскольку они являются функцией от решающих функций).

6.2 Контрольная выборка

Только что мы акцентировали внимание на том, что риск — случайная величина, но в данном разделе временно «забудем» про это. Здесь мы будем рассматривать оценку риска на основе контрольной выборки. В этом случае мы полностью оказываемся в рамках классической математической статистики, и риск будет

неизвестной величиной, но не случайной.

6.2.1 Точечная оценка

Пусть мы имеем некоторую решающую функцию $\lambda(x)$. При этом не важно, как эта функция получена: использовался некоторый метод и обучающая выборка, или же функция выбрана вообще «наугад». Считаем решающую функцию фиксированной. Тогда риск — фиксированная неизвестная величина.

Пусть $S^* = ((x^i, y^i) \in D \mid i = 1, \dots, N^*)$ — «новая» случайная независимая выборка из распределения P_c , $S^* \in D^{N^*}$.

Оценку риска определим как средние потери на контрольной выборке:

$$R^*(S^*, \lambda) = \frac{1}{N^*} \sum_{i=1}^{N^*} \mathcal{L}(y^i, \lambda(x^i)).$$

Здесь мы обозначили объём контрольной выборки за N^* , чтобы подчеркнуть, что он не обязан совпадать с N — объёмом обучающей выборки. Однако, чтобы не загромождать обозначения в дальнейшем размер контрольной выборки также будем обозначать как N .

В случае индикаторной функции потерь риск — это вероятность ошибочной классификации, а оценка на контрольной выборке — это просто доля ошибок на ней. Таким образом имеем классическую задачу оценивания вероятности события в схеме Бернулли на основе частоты этого события в эксперименте. «Событием» здесь является факт ошибочного прогноза.

Данная задача очень хорошо изучена. Известно, что частота события — это несмешённая, состоятельная и эффективная оценка вероятности. Например, факт состоятельности известен как «закон больших чисел».

6.2.2 Доверительный интервал

Рассмотрим вопрос, как практически использовать оценку по контрольной выборке. Например, на 10 контрольных объектах ре-

шающая функция не допустила ни одной ошибки. Что можно сказать о вероятности ошибки на новых объектах?

Наиболее естественным кажется в качестве оценки вероятности ошибки взять её частоту. Частота ошибок в данном примере равна 0. Но корректно ли будет утверждать, что вероятность ошибки для данной решающей функции равна 0? Очевидно, что нет. Если сделать такое заявление, то пользователи этой модели будут полностью ей доверять (т.е. считать, что она не ошибается). При этом понятно, что проверка модели на 10 объектах не может быть основанием для такой уверенности.

Ситуация в примере эквивалентна следующей задаче. Известно, что в урне белые и чёрные шары. Извлекли 10 шаров, все оказались белыми. Какой прогноз о цвете следующего шара?

Математически корректный ответ можно дать, построив доверительный интервал.

Пусть p – вероятность чёрного шара, N – число извлечённых шаров, M – количество чёрных среди них. Тогда

$$P_{p,N}(M) = C_N^M p^M \cdot (1-p)^{N-M}, \quad P_{p,N}(0) = (1-p)^N.$$

Если, например, $p = 0,5$, то $P_{p,N}(0) = (0,5)^{10} \approx 0,001$. Получается, что при таком значении p вероятность получить 10 белых шаров из 10 очень низкая, но это случилось. Значит предположение, что $p = 0,5$ является неправдоподобным и на самом деле p гораздо меньше.

При уменьшении p вероятность полученного в эксперименте результата будет увеличиваться и с какого-то момента станет достаточно большой, чтобы считать ситуацию правдоподобной.

Иными словами, идея построения доверительного интервала заключается в том, что мы отвергаем все варианты (гипотезы относительно значения оцениваемого параметра), при которых вероятность полученного в эксперименте результата «слишком низкая». В каком то смысле мы считаем если у события «слишком низкая» вероятность, то оно «не должно было» случиться.

Порог, начиная с которого мы считаем вероятность «слишком низкой», называется уровнем значимости α и выбирается эвристически, например 0,05. Также используется понятие доверительной вероятности, которая есть $\eta = 1 - \alpha$.

Положив $P_{p,N}(0) = \alpha$, имеем $p = 1 - \alpha^{\frac{1}{N}} = 1 - e^{\frac{\ln \alpha}{N}}$.

При $\alpha = 0,1$ и $N = 10$ получим $p \approx 0,2$.

Для наглядности мы рассмотрели случай $M = 0$. В общем случае доверительные интервалы в схеме Бернулли выглядят следующим образом.

Односторонний интервал $[0, \hat{p}]$ находится из уравнения

$$\sum_{i=0}^M C_N^i \hat{p}^i \cdot (1 - \hat{p})^{N-i} = \alpha.$$

Двусторонний интервал $[p_1, p_2]$ находится из уравнений

$$\sum_{i=0}^M C_N^i p_2^i \cdot (1 - p_2)^{N-i} = \sum_{i=M}^N C_N^i p_1^i \cdot (1 - p_1)^{N-i} = \frac{\alpha}{2}.$$

На рис. 6.2.2 изображены односторонние доверительные интервалы для разных α (слева) и двусторонний интервал при $\alpha = 0,1$ (справа). Кривые «Equal» и «Bayes» на левом графике отражают усреднённые оценки, которые будут вычислены ниже.

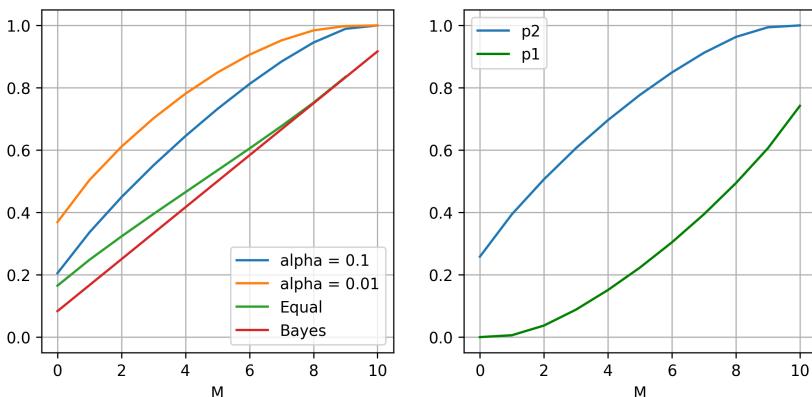


Рис. 6.1: Доверительные интервалы для риска при $N = 10$.

6.2.3 Усреднённые оценки

С математической точки зрения доверительный интервал для риска — это, пожалуй, наиболее полный ответ, который можно дать (оценивая риск). Однако для конечного пользователя модели такая конструкция (доверительный интервал для вероятности ошибки при заданной доверительной вероятности) является слишком сложной. Пользователю желательно получить одно число, которое он мог бы интерпретировать как ожидаемую вероятность ошибки для модели.

Это можно сделать, например, в рамках байесовского подхода, который является одним из классических подходов математической статистики при оценивании параметров.

Будем считать параметр случайной величиной и предположим, что она имеет равномерное априорное распределение, т.е. плотность вероятности $\varphi(p) \equiv 1$, $0 \leq p \leq 1$.

Тогда по формуле Байеса можем вычислить апостериорное распределение (при условии появления M успехов в серии N испытаний)

$$\varphi(p | M) = P_{p,N}(M) \frac{\varphi(p)}{P(M)}.$$

После преобразований, получаем

$$\varphi(p | M) = (N+1) C_N^M p^M \cdot (1-p)^{N-M}.$$

По условному распределению можем вычислить условное математическое ожидание

$$E_M p = \int_0^1 p \varphi(p | M) dp = \frac{M+1}{N+2}.$$

Получаем, что если модель на 10 контрольных объектах ни разу не ошиблась, то вероятность ошибки для неё следует оценить как $\frac{1}{12}$.

На рис. 6.2.2 эта оценка имеет метку «Bayes».

Байесовский подход имеет очевидный недостаток, связанный с необходимостью задавать априорное распределение, что в данном случае является никак не обоснованным произволом.

Попробуем действовать иначе. Заметим, что функция $\eta(\hat{p}) = 1 - \alpha(\hat{p})$ обладает всеми свойствами функций распределения (хоть ей и не является).

Однако чисто технически мы можем взять усреднение

$$\hat{\mathbb{E}}_{Mp} = \int_0^1 \hat{p} d\eta(\hat{p}) = \frac{M+1}{N+1}.$$

Данная операция выглядит сомнительно, но как известно: «если нельзя, но очень хочется, то — можно». Формально мы не назвали его математическим ожиданием, а вычислять вправе всё, что угодно. В результате мы получаем выражение, очень похожее на полученное из байесовского подхода. Выражение можно использовать при $M < \frac{N}{2}$.

Ещё один способ заменить доверительный интервал одним числом — это приравнять \hat{p} и α . Действительно, с уменьшением уровня значимости ширина доверительного интервала растёт, и есть точка, где они совпадают. Это значение также можно использовать как оценку риска. На рис. 6.2.2 эта оценка имеет метку «Equal».

6.2.4 Область применимости байесовского подхода

Данный раздел не имеет отношения к оцениванию риска, но поскольку мы только что коснулись байесовского подхода и больше не будем к нему возвращаться, обсудим этот подход подробнее.

Рассмотрим пример, который известен как «парадокс конвертов».

Игроку предлагается выбрать один из двух одинаковых на вид запечатанных конвертов с деньгами, причём известно, что сумма в одном из них в 10 раз больше, чем в другом. При этом игроку разрешается вскрыть один конверт, после чего решить, забрать его или оставшийся запечатанным.

Пусть в первом конверте оказалось x рублей.

Если считать, что во втором конверте может быть равновероятно $10x$ или $\frac{x}{10}$, то математическое ожидание выигрыша при

выборе второго конверта будет $5.05x$. Получается, что выгоднее взять второй конверт.

При этом решение не зависит от суммы x , которую мы обнаружили в первом конверте. Получается, что конверт мы могли и не вскрывать, а сразу взять второй конверт. Но если мы не вскрыли конверт, то конверты остаются равноправными, и не может один из них быть выгоднее другого. Имеем противоречие.

Парадокс является классическим примером некорректного использования байесовского подхода.

В отличие от разнообразных псевдопарадоксов (например «парадокса» Монти-Холла), которые просто отражают некоторые факты, непривычные для обывательского восприятия, парадокс конвертов представляется истинным парадоксом.

Наиболее известным истинным парадоксом является парадокс Рассела, который показывает, что некорректно рассматривать множества, которые могут включать само себя.

Таким образом, истинный парадокс демонстрирует некорректность некоторой математической конструкции.

Парадокс конвертов показывает, что вероятность — это объективная неопределённость (современная физика придерживается этой же позиции) и её нельзя использовать как «меру незнания».

Получается, что байесовский подход (а мы именно его использовали в рассуждениях с конвертами) может давать некорректные результаты. Тем не менее, его вполне можно использовать (с разумной осторожностью и осознавая возможные последствия).

При этом во многих случаях байесовский подход можно заменить подходящим образом введённой регуляризацией [Воронцов].

6.2.5 Случай произвольной функции потерь

В целях упрощения и наглядности, мы большую часть рассуждений вели для случая задачи классификации с индикаторной функцией потерь.

В общем случае выкладки как правило аналогичны.

Если критерий качества представляется математическим ожиданием функции потерь, но мы имеем классическую задачу стати-

стки, а именно оценивание математического ожидания случайной величины при неизвестном (чаще всего) распределении.

Для сокращения размера формул обозначим $\zeta_i = \mathcal{L}(y^i, \lambda(x^i))$ – величину потерь на i -м объекте контрольной выборки. Поскольку элементы выборки независимы, случайные величины ζ_i также независимы. При этом они имеют одинаковое распределение.

Требуется оценить математическое ожидание $\mu = E\zeta_i$, т.е. риск.

Используемая нами оценка R^* в текущих обозначениях принимает вид

$$\bar{\mu} = \frac{1}{N} \sum_{i=1}^N \zeta_i.$$

Чтобы оценить точность этой оценки, нужно оценить дисперсию $\sigma^2 = D\zeta_i$.

Рассмотрим оценку

$$\bar{\sigma}^2 = \frac{1}{N} \sum_{i=1}^N (\zeta_i - \bar{\mu})^2 \equiv \frac{1}{N} \sum_{i=1}^N \zeta_i^2 - \bar{\mu}^2.$$

Вычислим её математическое ожидание

$$E\bar{\sigma}^2 = E \frac{1}{N} \sum_{i=1}^N \zeta_i^2 - E\bar{\mu}^2 = E\zeta_i^2 - E\bar{\mu}^2.$$

$$E\bar{\mu}^2 = \frac{1}{N^2} E \left(\sum_{i=1}^N \zeta_i \right)^2 = \frac{1}{N^2} \left(\sum_{i=1}^N E\zeta_i^2 + \sum_{j \neq i} E\zeta_i \zeta_j \right).$$

Здесь мы раскрыли квадрат суммы и сгруппировали слагаемые. В силу независимости имеем $E\zeta_i \zeta_j = E\zeta_i E\zeta_j = \mu^2$. Откуда

$$E\bar{\mu}^2 = \frac{1}{N} E\zeta_i^2 + \frac{N-1}{N} \mu^2.$$

Наконец

$$E\bar{\sigma}^2 = \frac{N-1}{N} (E\zeta_i^2 - \mu^2) = \frac{N-1}{N} \sigma^2.$$

Мы вывели известный факт, что несмешённая оценка дисперсии есть

$$\tilde{\sigma}^2 = \frac{N}{N-1}\bar{\sigma}^2.$$

Можно было вывести это же на основе того, что дисперсия суммы независимых величин равна сумме дисперсий. Из этого следует

$$D\bar{\mu} = \frac{1}{N}\sigma^2$$

С другой стороны

$$D\bar{\mu} \equiv E\bar{\mu}^2 - (E\bar{\mu})^2 = E\bar{\mu}^2 - \mu^2.$$

Откуда

$$E\bar{\mu}^2 = \mu^2 + \frac{1}{N}\sigma^2.$$

Снова получаем

$$E\bar{\sigma}^2 = E\zeta_i^2 - \mu^2 - \frac{1}{N}\sigma^2 = \sigma^2 - \frac{1}{N}\sigma^2.$$

Подобный вывод приводится во всех курсах по математической статистике и достаточно было сослаться на них, но мы подробно воспроизвели выкладки ввиду их важности.

Итак, $\tilde{\sigma}^2$ является несмешённой оценкой для σ^2 , а $D\bar{\mu} = \frac{1}{N}\sigma^2$, поэтому $\frac{1}{N}\tilde{\sigma}^2$ будет несмешённой оценкой для $D\bar{\mu}$.

На основе полученных выражений можно построить доверительный интервал. Введём величину

$$t = (\bar{\mu} - \mu) \frac{\sqrt{N}}{\tilde{\sigma}}.$$

Если ζ_i имеют нормальное распределение, то случайная величина t имеет распределение Стьюдента с $N-1$ степенями свободы, которое с ростом N сходится к нормальному распределению. При реально используемых размерах выборки распределение можно заменить нормальным.

В реальных задачах ζ_i крайне редко имеют нормальное распределение. Однако в любом случае в силу центральной предельной теоремы распределение величины $\bar{\mu}$ сходится кциальному.

На практике это означает, что если мы построили несмешённую оценку для дисперсии средних потерь на контрольной выборке, то можем использовать её для оценки доверительного интервала, в том числе по правилу «три сигма».

Заметим, что скорость сходимости к нормальному закону существенно замедляется при наличии «выбросов», или «тяжёлых хвостов» у распределения величин ζ_i . В таких случаях доверительный интервал, построенный на нормальном приближении, может иметь значительную погрешность.

Для критериев, не представляемых через функцию потерь (например ROC-AUC) доверительный интервал построить сложнее, однако задача так или иначе решаема (возможно, средствами статистического моделирования).

Предположим теперь, что мы разбили контрольную выборку на K частей, которые будем называть фолдами. Пусть J_k – множество индексов объектов, принадлежащих k -му фолду, а N_k – размер фолда.

Обозначим

$$\bar{\mu}_k = \frac{1}{N_k} \sum_{i \in J_k} \zeta_i.$$

Для удобства будем считать, что N делится на K . Положим все N_k одинаковыми, т.е. $N_k \equiv \frac{N}{K}$.

Тогда все μ_k будут независимыми одинаково распределёнными случайными величинами со средним μ и дисперсией

$$\sigma_K^2 = \frac{\sigma^2}{N_k} = \sigma^2 \frac{K}{N}.$$

Оценка

$$\tilde{\sigma}_K^2 = \frac{1}{K-1} \sum_{k=1}^K (\bar{\mu}_k - \bar{\mu})^2$$

будет несмешённой оценкой для σ_K^2 , а значит $\hat{\sigma}_K^2 = \tilde{\sigma}_K^2 \cdot \frac{N}{K}$ будет несмешённой оценкой для σ^2 .

Означает ли это, что оценку $\hat{\sigma}_K^2$ можно использовать наряду с $\tilde{\sigma}^2$ в качестве оценки параметра σ^2 ?

Формально использовать можно, но это будет очень плохая оценка.

Чтобы убедиться в этом, вычислим дисперсию этой оценки. Поскольку в общем случае это сделать не получится, предположим, что ζ_i имеют нормальное распределение.

Тогда $(K - 1) \cdot \frac{N\tilde{\sigma}_K^2}{K\sigma^2}$ имеет распределение χ^2 с $K - 1$ степенями свободы. Дисперсия такого распределения известна и равна $2(K - 1)$.

Откуда

$$D\hat{\sigma}_K^2 = 2(K - 1) \left(\frac{\sigma^2}{K - 1} \right)^2 = \frac{2\sigma^4}{K - 1}.$$

Минимальная дисперсия оценки достигается при $K = N$, т.е. при негруппированной выборке. Это вполне естественно: если мы хотим оценить дисперсию, т.е. «колебания» значений, то не следует их сглаживать усреднением. Использование такой оценки эквивалентно по точности уменьшению объема выборки до K .

Оценка

$$\check{\sigma}_K^2 = \frac{1}{N - K} \sum_{j=1}^N (\zeta_j - \bar{\mu}_{k(j)})^2$$

также является несмешённой оценкой для σ^2 . Здесь $k(j)$ – номер фолда, в который попадает j -й объект.

Если ζ_i имеют нормальное распределение, то $(N - K) \cdot \frac{\check{\sigma}_K^2}{\sigma^2}$ имеет распределение χ^2 с $N - K$ степенями свободы. Из этого можно вывести, что

$$D\check{\sigma}_K^2 = \frac{2\sigma^4}{N - K}.$$

Таким образом, точность оценки $\check{\sigma}_K^2$ снижается с ростом числа фолдов так, будто выборка уменьшается на K .

То, что группировка выборки уменьшает точность оценивания дисперсии, было вполне очевидно, и приведённые выкладки кажутся излишними. Действительно, в случае контрольной выборки данные манипуляции не имеют практического смысла, однако разбиение на фолды будет в дальнейшем использоваться при

кросс-валидации, и тогда выведенные сейчас формулы нам пригодятся.

6.2.6 Свойства оценки по контрольной выборке

Средний риск на контрольной выборке как оценка риска обладает следующими свойствами:

- простота вычисления,
- несмешённость (не совсем в том смысле, в котором хотелось бы),
- состоятельность,
- известен точный доверительный интервал,
- эффективность (не в том смысле, в котором хотелось бы),
- требует дополнительной выборки.

Некоторые свойства, очевидно, требуют пояснений. Например, почему несмешённость оценки не такая, которую бы хотелось иметь. Вспомним, что получив ноль ошибок на контрольной выборке, мы оценили риск ненулевым значением. Получается, что вместо изначально несмешённой оценки мы использовали скорректированную оценку, которая формально является смешённой. Тем не менее, несмешённость — важное свойство оценки.

Что касается эффективности, то формально эффективность имеет место. А неформально эффективность интерпретируется как то, что оценка в некотором смысле самая лучшая.

При условии использования контрольной выборки и неиспользования обучающей, рассмотренную оценку действительно можно считать самой лучшей. Но на практике идеальным вариантом было бы обойтись вообще без контрольной выборки (использовать обучающую). Получается, что эффективность в данном случае не означает, что оценка лучшая из возможных.

В качестве общего вывода можно сказать, что контрольная выборка — единственный на настоящий момент способ получить

оценку риска с математически строгим доверительным интервалом (т.е. с известной точностью).

Для двух случаев, а именно для биномиального и для нормального распределения риска, мы привели выражения для точных доверительных интервалов. В остальных случаях можно пользоваться нормальным приближением либо статистическим моделированием.

6.3 Эмпирический риск

В данном разделе мы «вспомним», что решающая функция строится по выборке, а значит риск — случайная величина.

Пусть $S = \{(x^i, y^i) \in D \mid i = 1, \dots, N\}$ — случайная независимая выборка из распределения P_c , $S \in D^N$.

Эмпирический риск определим как средние потери на обучающей выборке:

$$\tilde{R}(S, \lambda) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(y^i, \lambda(x^i)).$$

6.3.1 Смещённость эмпирического риска

Чтобы понять причины смещённости эмпирического риска, рассмотрим простой пример.

Предположим, что мы производим некоторые изделия и хотим убедить клиентов, что продукция имеет очень низкую долю брака. Для этого мы берём партию из $N = 100$ изделий и проверяем её.

Предположим, что обнаружилось 5 бракованных изделий. Тогда мы должны объявить клиентам, что наша продукция имеет приблизительно 5% брака.

Однако нас это не устраивает и мы проверяем следующую партию из $N = 100$ изделий. В ней оказывается 6 бракованных изделий. Проверяем дальше, пока, наконец, в 198-й партии не оказываются все изделия качественными. Публикуем видео с процессом проверки этой партии и объявляем, что наша продукция имеет 0% брака.

Легко посчитать, что если вероятность выпуска бракованного изделия составляет 0,05, то вероятность получить партию из 100 исправных изделий составляет 0,006. При этом вероятность получить из 200 таких партий хотя бы одну, состоящую только из исправных изделий, составляет уже 0,7.

Получается, что реальная доля брака в продукции, скорее всего, и составляет 5%, а то, что нам удалось (за 198 попыток) найти партию без брака, этому не противоречит. Таким образом, оценка, что доля брака составляет 0%, является смещённой, т.е. систематически заниженной. Это вполне очевидно, поскольку мы специально делали «подгонку» под нужный результат.

Практически такая же по смыслу «подгонка» имеет место в машинном обучении.

Предположим, что мы строим линейный классификатор. Если бы мы выбрали разделяющую плоскость наугад, что эмпирический риск был бы несмещённой оценкой риска. Но поскольку мы из всех гиперплоскостей выбираем именно ту, которая лучше свого разбьёт обучающую выборку, происходит «подгонка», и эмпирический риск становится смещённым.

Однако, пример выше мы привели не только для того, чтобы проиллюстрировать причину смещённости эмпирического риска, но и для того, чтобы показать, что даже по эмпирическому риску возможно оценивать риск.

Для этого поставим себя на место потенциального клиента, который увидел ролик с исправными 100 приборами, но при этом узнал, что для этого ролика выбрана лучшая из 200 партий изделий.

Заметим, что если вероятность брака равна 0,08, то вероятность получить 100 исправных в лучшей из 200 партий составляет менее 0,05, т.е. меньше типичного уровня значимости, используемого при проверке гипотез. Исходя из этого, гипотезу, что вероятность брака больше 0,08, мы можем отвергнуть.

Таким образом, даже на основе смещённой оценки удаётся получить содержательную оценку доверительного интервала для вероятности брака. Для этого потребовалось знать «число попыток» (число партий).

Аналогичная идея используется при получении оценок Вапника-Червоненкиса.

6.3.2 Оценки Вапника-Червоненкиса

Повторим рассуждения, но вместо деталей будут объекты выборки, а вместо брака — факт ошибочной классификации.

Обозначим через p вероятность ошибочной классификации, которая нам не известна и которую мы будем оценивать.

Пусть N_e — число ошибочно классифицированных объектов выборки объёма N , обозначим долю ошибок как $\nu = \frac{N_e}{N}$.

Оценим вероятность того, что ν будет отличаться от p более чем на ε .

Как известно, N_e подчиняется биномиальному распределению с параметром p . В силу центральной предельной теоремы распределение величины ν может быть приближено нормальным распределением с математическим ожиданием p и дисперсией $\frac{p(1-p)}{N}$. Имеем

$$\mathbb{P}(|\nu - p| \geq \varepsilon) \approx 1 - \Phi\left(\varepsilon \sqrt{\frac{N}{p \cdot (1-p)}}\right) \leq 1 - \Phi\left(2\varepsilon\sqrt{N}\right),$$

где $\Phi(z) = \frac{1}{\sqrt{2\pi}} \int_{-z}^z e^{-\frac{t^2}{2}} dt$.

Мы воспользовались фактом $\max_{0 \leq p \leq 1} \sqrt{p \cdot (1-p)} = \frac{1}{2}$.

Интеграл от нормальной плотности не выражается аналитически

$$1 - \Phi(z) = 2 \frac{1}{\sqrt{2\pi}} \int_z^\infty e^{-\frac{t^2}{2}} dt,$$

но мы его можем оценить:

$$\begin{aligned} \int_z^\infty e^{-\frac{t^2}{2}} dt &= \int_0^\infty e^{-\frac{(u+z)^2}{2}} du = \int_0^\infty e^{-\frac{u^2+2uz+z^2}{2}} du \leq \\ &e^{-\frac{z^2}{2}} \int_0^\infty e^{-uz} du = \frac{1}{z} e^{-\frac{z^2}{2}}. \end{aligned}$$

По ходу выкладок мы сделали замену $t = u + z$ и воспользовались тем, что $e^{-\frac{u^2}{2}} \leq 1$.

В результате при $z > 1$ имеем

$$1 - \Phi(z) \leq 2 \frac{1}{z\sqrt{2\pi}} e^{-\frac{z^2}{2}} < e^{-\frac{z^2}{2}}.$$

Если $2\varepsilon\sqrt{N} > 1$, то

$$\mathsf{P}(|\nu - p| \geq \varepsilon) < e^{-2\varepsilon^2 N}.$$

Мы оценили для заданной решающей функции вероятность того, что эмпирический риск будет отличаться от вероятности ошибки больше чем на ε .

Оценим теперь вероятность того, что эмпирический риск будет отличаться от вероятности ошибки больше чем на ε хотя бы для одной решающей функции, если всего функций L . Эта вероятность больше предыдущей, но не более чем в L раз, поскольку вероятность суммы событий не превосходит сумму вероятностей. Поэтому

$$\mathsf{P}\left(\bigvee_{\lambda \in \Lambda} |\nu(\lambda) - p(\lambda)| \geq \varepsilon\right) < L e^{-2\varepsilon^2 N}.$$

Приведённые выкладки аналогичны тому как мы оценивали вероятность получения серии изделий без брака хотя бы в одной из множества партий.

Нам нужно, чтобы вероятность выхода за окрестность была достаточно маленькой. Пусть эта вероятность равна $1 - \eta$, где η – выбранная доверительная вероятность (которая равна, например, 0,95). Приравняем полученную оценку к этой вероятности. Имеем $L e^{-2\varepsilon^2 N} = 1 - \eta$

$$\ln L - 2\varepsilon^2 N = \ln(1 - \eta), \quad \varepsilon = \sqrt{\frac{\ln L - \ln(1 - \eta)}{2N}} \approx \sqrt{\frac{\ln L}{2N}}.$$

Как и в законе больших чисел, отклонение обратно пропорционально корню из объёма выборки. Однако в числителе теперь стоит логарифм числа решающий функций.

6.3.3 Функция роста

Практически все реально используемые классы решающих функций бесконечны. Например, мощность множества всех гиперплоскостей — континuum.

Чтобы применить оценки предыдущего подраздела в такой ситуации, нужно вместо количества функций взять количество классов эквивалентности. Эквивалентными мы считаем функции, которые одинаково классифицируют точки выборки.

Теперь L будет не число решающих функций, а число способов классифицировать выборку функциями класса.

Максимальное число способов приписать различные классы N объектам составляет 2^N . Но не всегда все эти варианты реализуемы функциями выбранного класса.

Например, 4 точки (в общем положении) в пространстве можно плоскостью разбить на 2 множества всеми способами, а 5 точек — уже не всеми.

Если точки лежат не в общем положении (например 4 точки в пространстве оказались на одной плоскости), то число способов разбить такую выборку будет меньше.

В любом случае, при каждом N существует максимальное по всем выборкам этого объёма число способов классифицировать выборку на 2 класса. Это число называется функцией роста, которую будем обозначать \hat{L}_N

Для гиперплоскостей в n -мерном пространстве функция роста имеет следующее выражение

$$\hat{L}_N = 2 \sum_{i=0}^n C_{N-1}^i.$$

Ёмкостью h класса решающих функций называется максимальный размер выборки, которая может быть классифицированная этими функциями всеми 2^N способами. Таким образом, для линейных функций ёмкость равна $h = n + 1$.

Говорят, что класс имеет бесконечную ёмкость, если $\hat{L}_N = 2^N$ для всех N .

При $N > h$ для функции роста в общем случае справедлива следующая оценка

$$\hat{L}_N \leq \sum_{i=0}^h C_N^i \leq 1,5 \cdot \frac{N^h}{h!}.$$

Отсюда следует, что $\ln L$ может быть оценен сверху величиной порядка $h \ln N$, из чего заключаем, что для классов конечной ёмкости эмпирический риск является состоятельной оценкой риска.

Приведённый результат важен с теоретической точки зрения, однако практическая значимость его ограничена.

Во-первых, методы с конечной ёмкостью класса решений сейчас применяются существенно реже, чем так называемые универсальные методы, в которых сложность решения подстраивается под размер выборки. Таким свойством обладают деревья решений и построенные на основе них ансамбли.

Во-вторых, близость эмпирического риска к риску сама по себе не имеет практического значения. Иными словами, нам не важно, насколько близко качество решения на обучении и на контроле (на новых объектах). На практике важно лишь само значение риска на новых объектах.

6.3.4 Неулучшаемость оценок

Данный раздел имеет повышенную сложность и при первом чтении может быть пропущен.

Будем рассматривать случай конечного множества классификаторов.

Пусть p — вероятность «успеха» в схеме Бернулли. Для фиксированного классификатора в роли p будет выступать вероятность ошибочной классификации.

Обозначим через $\xi = \frac{N_e}{N}$ случайную величину, представляющую собой долю ошибочно классифицированных объектов обучающей выборки,

$$B(\gamma, N, p) = P(\xi \leq \gamma) = \sum_{0 \leq i \leq N\gamma} C_N^i p^i (1-p)^{N-i}$$

— кумулятивное биномиальное распределение.

Имеем: $P(\xi \leq \gamma) = B(\gamma, N, p)$ — вероятность получить долю ошибочно классифицированных объектов (эмпирический риск) меньшее γ .

Если приравнять данную вероятность заданному уровню значимости α , то получим уравнение, связывающее p и γ . Выразив p как функцию γ , получим границу доверительного интервала для вероятности ошибочной классификации.

Пусть $\hat{p}(\gamma)$ — функция, задаваемая уравнением

$$B(\gamma, N, \hat{p}(\gamma)) = \alpha.$$

Очевидно, что для любого p выполняется $P(p > \hat{p}(\xi)) \leq \alpha$.

Рассмотрим конечное множество классификаторов Λ , $|\Lambda| = \mathcal{C}$.

Для каждого $\lambda \in \Lambda$ определена вероятность ошибочной классификации $p(\lambda)$.

Обозначим $A(\lambda)$ — событие $p(\lambda) > \hat{p}(\gamma)$.

Имеем:

$$\begin{aligned} P(p(\lambda(\nu)) > \hat{p}(\gamma)) &\leq \\ P\left(\sum_{\lambda \in \Lambda} A(\lambda)\right) &\leq \sum_{\lambda \in \Lambda} P(A(\lambda)) \leq \alpha \mathcal{C}. \quad (6.1) \end{aligned}$$

Таким образом, при доверительной вероятности $\eta = 1 - \alpha \mathcal{C}$ функция $\hat{p}(\gamma)$ является доверительным интервалом для вероятности ошибочной классификации при любом методе $\lambda(\nu)$ выбора решающей функции из Λ . Получили один из вариантов оценок Вапника-Червоненкиса.

В оценке присутствуют три неравенства. На первый взгляд может показаться, что первое из них несущественно влияет на точность оценки для метода минимизации эмпирического риска по всему классу Λ , поскольку при минимизации эмпирического риска, как правило, максимизируется разность между риском и эмпирическим риском. Однако классификатор, минимизирующий эмпирический риск, не обязательно оказывается среди нарушающих доверительный интервал. Фактически это есть эффект «расслоения» [11]. Помимо «расслоения», погрешность оценки этим неравенством может быть вызвана тем, что метод классификации не

всегда минимизирует эмпирический риск (использует более сложный критерий качества или приближенный метод оптимизации).

Второе неравенство (известное также как «union bound») обращается в равенство только в случае, когда вероятность произведения любой пары событий равна нулю. В случае независимых событий, вероятность которых отлична от 0 и 1, неравенство становится строгим. Однако его вклад в погрешность оценок в этом случае является несущественным. Так, например, если A_j — независимы (в совокупности) и $P(A_j) = p$, то

$$P\left(\sum_{j=1}^C A_j\right) = 1 - (1-p)^C \xrightarrow[C \rightarrow \infty]{pC=const} 1 - e^{-pC}.$$

При $1 - e^{-pC} = 0,2$ получим $pC \approx 0,22$.

Случай независимых событий представляет собой пример, когда неравенство union bound не вносит существенной погрешности в оценку риска, и именно на независимых событиях строится пример, показывающий, что сложностная оценка является асимптотически точной (рассмотрен далее). Вместе с тем, в большинстве практических ситуаций события $A(\lambda)$ зависимы, причем зависимость такова, что погрешность union bound становится весьма значительной. Это эффект «сходства» классификаторов [11].

Заметим, что эффекты «расслоения» и «сходства» связаны между собой, в частности, для независимых $A(\lambda)$ эффект «расслоения» также не имеет места (далее иллюстрируется на примере).

Последнее неравенство в оценке (6.1) есть следствие дискретности биномиального распределения и не дает существенного вклада в погрешность.

Простейшие примеры

Пусть $C = 1$. Оценка риска превращается в классический односторонний доверительный интервал для вероятности успеха в схеме Бернулли. Следующий пример: $C = 2$, причем λ_2 является инверсией λ_1 , т. е. $\lambda_2(x) = 1 - \lambda_1(x)$. Можно для определенности считать, что первый классификатор всегда приписывает класс 0, второй — класс 1. Это соответствует методу классификации объектов по оценкам безусловных вероятностей классов (то есть в соответствии с выборочными частотами классов, без использования

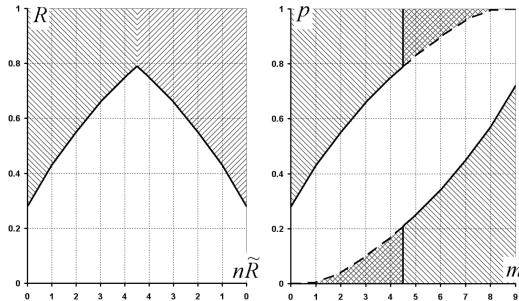


Рис. 6.2: Критические области.

признакового пространства X). Вероятностная модель задается единственным параметром $p = P(y = 0)$, и пусть m — количество объектов с $y = 0$ в выборке.

Оценка по формуле (6.1) совпадает с оценкой, полученной из двустороннего доверительного интервала для вероятности успеха в схеме Бернулли. Однако такой интервал неоптимален, поскольку включает в критическое множество лишние области (см. рис. 7.1.2). За счет «расслоения» часть критической области двустороннего доверительного интервала не входит в критическую область оценки риска. Это происходит, когда выход за границу двустороннего интервала происходит на классификаторе, который имеет не наилучший эмпирический риск. На рисунке эта область показана двойной штриховкой (критические множества — одинарной).

Асимптотические оценки

Будем рассматривать асимптотическое поведение оценок (6.1), когда абсолютный объем выборки стремится к бесконечности, но при этом относительно сложности остается малым [12], т. е. $N \rightarrow \infty$, $\varkappa = \frac{N}{\ln C} = const$.

Хотя рассматривается асимптотический случай, полученные выводы будут на качественном уровне применимы и к случаю достаточно малых выборок $N \approx 50$. Рассмотрим простейший вариант сложностной оценки [9]: $\hat{p}(\gamma) = \gamma + \varepsilon$, где

$$\varepsilon = \sqrt{\frac{\ln C - \ln(1 - \eta)}{2N}} = \sqrt{\frac{1}{2\varkappa} \left(1 + \frac{\varkappa}{N} \ln(1 - \eta) \right)}.$$

При $\varkappa \ll N$, что, как правило, выполняется на практике, оценка определяется главным образом значением \varkappa , в то время как N и η несущественны. Например, при $\varkappa = 5$ и $N = \infty$, имеем $\varepsilon = 0,316$, при $N = 50$ и $\eta = 0,5$ получим $\varepsilon = 0,327$, при $\eta = 0,9$ получим $\varepsilon = 0,351$.

Таким образом, при типичных значениях параметров оценка доверительного интервала слабо зависит от доверительной вероятности (если не выбирать значения близкие к 1), а объем выборки влияет, главным образом, опосредованно через \varkappa . Аналогичные рассуждения можно провести и для более тонких сложностных оценок. Это оправдывает рассмотрение предложенной асимптотики, позволяющей ограничиться наиболее существенным параметром — \varkappa .

Чтобы построить доверительный интервал риска, пользуясь (6.1), нужно найти $\hat{p}(\gamma)$, удовлетворяющую уравнению

$$\mathcal{CB}(\gamma, N, \hat{p}(\gamma)) = 1 - \eta. \quad (6.2)$$

Справедлива известная [13] оценка

$$\begin{aligned} \frac{1}{\sqrt{2\pi N\gamma(1-\gamma)}} \exp\left(-NH(\gamma, p) - \frac{1}{12N\gamma(1-\gamma)}\right) &\leqslant \\ &\leqslant B(\gamma, N, p) \leqslant \exp(-NH(\gamma, p)), \end{aligned}$$

где $H(\gamma, p) = \gamma \ln \frac{\gamma}{p} + (1 - \gamma) \ln \frac{1-\gamma}{1-p}$ — дивергенция Кульбака-Лейблера.

Умножая неравенства на \mathcal{C} , получаем, что решения уравнения (6.2) при $N \rightarrow \infty$, $\varkappa = \frac{N}{\ln \mathcal{C}}$ сходятся к решению уравнения

$$H(\gamma, \hat{p}(\gamma)) = 1/\varkappa. \quad (6.3)$$

Решение $\hat{p}_\varkappa(\gamma)$ уравнения (6.3) оказывается неулучшаемой (без использования дополнительной информации) асимптотической оценкой риска на основе эмпирического риска.

Для доказательства этого факта рассмотрим пример, приведенный в [14]. Пусть дан набор переменных X_1, \dots, X_C и множество классификаторов $\lambda_1, \dots, \lambda_C$, причем λ_j приписывает объекту класс, номер которого равен значению j -й переменной, то есть

$f_{\lambda_j}(x) = x_j$. Распределение в D задается следующим образом:

$$\begin{aligned}\mathsf{P}(x, y) &= \mathsf{P}(x | y) \mathsf{P}(y); \quad \mathsf{P}(x | y) = \sum_{j=1}^C \mathsf{P}(x_j | y); \\ \mathsf{P}(y = 0) &= p_0; \quad \mathsf{P}(x_j \neq y | y) = p,\end{aligned}$$

где p_0 и p — параметры распределения, причем выбор p_0 не имеет значения. По построению, риск для любого классификатора λ_j равен p . Вместо (6.1) для алгоритма $\lambda(\nu)$, минимизирующего эмпирический риск, в рассмотренной модели можем выписать точную вероятность выхода из доверительного интервала:

$$\mathsf{P}(p(\lambda(\nu)) > \hat{p}(\gamma)) = 1 - (1 - B(\gamma, N, p))^C = 1 - \eta.$$

Аппроксимируя степень экспонентой, получаем:

$$CB(\gamma, N, \hat{p}(\gamma)) \approx -\ln \eta.$$

От (6.2) данное уравнение отличается только правой частью. Но, так как асимптотическое решение не зависит от правой части, получаем, что для полученного уравнения решение есть $\hat{p}_*(\gamma)$. Если взять любую $\hat{p}(\gamma)$, строго меньшую $\hat{p}_*(\gamma)$ на некотором, ненулевой длины, интервале значений аргумента, то найдутся такие параметры рассмотренной модели, что вероятность выхода из доверительного интервала $\hat{p}(\gamma)$ будет больше любого $1 - \eta < 1$. Это доказывает, что оценка $\hat{p}_*(\gamma)$ является асимптотически неулучшаемой.

То, что без дополнительной информации нельзя добиться существенного улучшения оценок Вапника-Червоненкиса, общеизвестно. В данном разделе показано, что в рассмотренной асимптотике они неулучшаемы вовсе.

6.3.5 Точные асимптотические оценки для гистограммного классификатора

Пусть $X = \{1, \dots, k\}$ — единственная дискретная переменная, и метод классификации минимизирует эмпирический риск в каждой точке.

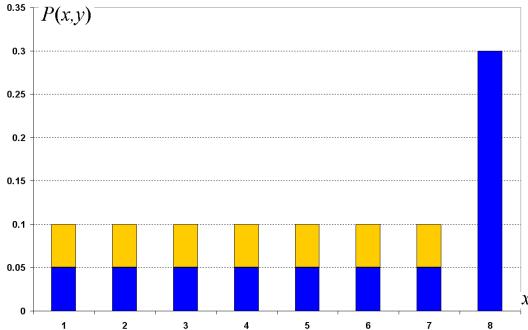


Рис. 6.3: «Худшее» распределение.

Вероятностная мера c задается набором вероятностей $\zeta_j = \mathbb{P}(x=j)$, $p_j = \mathbb{P}(y=0|x=j)$.

Данный пример имеет практическую ценность — он соответствует использованию histogram classifier, когда решение принимается по гистограммам частот [15].

В асимптотике доверительный интервал совпадает с $\widehat{S} + \widetilde{R}$, где $\widehat{S} = \sup_{c \in C} \mathbb{E}R - \mathbb{E}\widetilde{R}$ — функция максимального (по всем распределениям) смещения эмпирического риска [12].

В рассмотренном случае существенную роль играет эффект «сходства» классификаторов. Их множество продуцирует всего $\log_2 \mathcal{C} = k$ «ячеек», в то время как максимально возможное число «ячеек» (имеющее место в примере, иллюстрирующем достижимость сложностных оценок) составляет $2^{\mathcal{C}}$. Под «ячейками» здесь понимаются области порождаемого классификаторами разбиения признакового пространства. Число таких областей представляется перспективной для дальнейшего исследования характеристикой сложности семейства классификаторов, которая может использоваться наряду с ёмкостью.

6.3.6 Свойства эмпирического риска

Для оценки эмпирического риска можно отметить следующие свойства:

- простота вычисления,

- сильная смещённость,
- состоятельность при соответствующем ограничении на сложность метода (оценки Вапника-Червоненкиса),
- малая дисперсия.

Все они подробно обсуждались и не требуют дополнительных комментариев.

Следует сделать общий вывод относительно области применения эмпирического риска.

Несмотря на то, что на его основе мы не знаем способа с приемлемой точностью оценить риск (за редким исключением), вычисление эмпирического риска имеет смысл в качестве верхней оценки качества, а также для контроля процесса обучения.

При использовании методов малой ёмкости (например, линейных или наивного байесовского) при большом объёме выборки смещение эмпирического риска становится незначительным, и его можно использовать непосредственно как оценку риска.

6.4 Скользящий экзамен

Использование контрольной выборки требует дополнительных данных, а эмпирический риск, как правило, сильно смещён. Однако есть способ избежать этих недостатков.

Идея скользящего экзамена заключается в том, чтобы использовать всю выборку и для обучения, и для контроля, но не одновременно, а так, чтобы объекты, на которых проводится тестирование, не участвовали в обучении.

6.4.1 Определение

Функционал скользящего экзамена определяется как

$$\check{R}(S, Q) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(y^i, \lambda_{Q, S'_i}(x^i)),$$

где $S'_i = S \setminus (x^i, y^i)$ — выборка, получаемая из S удалением i -го наблюдения.

Суть процедуры в том, что каждый объект по очереди удаляется из обучающей выборки, по оставшимся объектам строится решающая функция, по которой делается прогноз для удалённого (отложенного) объекта. В результате мы получаем прогнозы для всех объектов. Далее для прогнозов вычисляем потери и усредняем.

В англоязычной терминологии такая процедура называется leave-one-out.

Данная процедура достаточно трудоёмкая вычислительно, поскольку требует провести обучение N раз.

Для сокращения затрат можно удалять объекты не по одному, а группами. Тогда получим метод кроссвалидации (K -fold cross-validation).

Исходная выборка разбивается на K равных частей, которые называются фолдами. Оценка определяется как

$$\check{R}^K(S, Q) = \frac{1}{N} \sum_{i=1}^N \mathcal{L}(y^i, \lambda_{Q, S'_{k(i)}}(x^i)),$$

где $S'_{k(i)}$ — выборка, получаемая из S удалением всей подвыборки (всего фолда), которой принадлежит i -е наблюдение. Здесь $k(i)$ — номер фолда, которому принадлежит i -й элемент выборки.

Для простоты выкладок полагаем, что N кратно K . В общем случае фолды будут не совсем равными, что практически несущественно.

Заметим, что в некоторых источниках такая оценка называется *out-of-fold*, а собственно кроссвалидацией называется эта же сумма, но со сгруппированными слагаемыми. Однако простая группировка слагаемых не кажется достаточным основанием для введения отдельных терминов, поэтому термин *out-of-fold* мы использовать не будем.

Оценка leave-one-out является частным случаем K -fold cross-validation при $K = N$.

Замечание по терминологии. Часто под скользящим экзаменом понимается именно leave-one-out. Мы же будем использовать термин «скользящий экзамен» двояко: в узком смысле как leave-one-out, и в широком смысле как любой метод, использующий всю выборку и для обучения, и для контроля, но так, чтобы объекты, на которых проводится тестирование, не участвовали в обучении.

Для дальнейших рассуждений нам нужно ввести понятие hold-out оценки. Она определяется как

$$\check{R}_k(S, Q) = \frac{1}{N_k} \sum_{i \in J_k} \mathcal{L}(y^i, \lambda_{Q, S'_k}(x^i)),$$

где N_k — размер k -го фолда, а J_k — множество индексов, ему принадлежащих.

Отличие от кроссвалидации только в том, что перебор фолдов не происходит, и только один фолд выступает в роли тестового.

Очевидно, что

$$\check{R}^K(S, Q) = \frac{1}{N} \sum_{k=1}^K N_k \cdot \check{R}_k(S, Q).$$

Кажется, что оценка hold-out совпадает с оценкой на основе тестовой выборки. Однако между ними есть важное отличие. Тестовую выборку мы используем для оценки качества финального

решения. А процедура hold-out подразумевает, что финальное решение мы будем строить заново, обучаясь уже по всей выборке.

6.4.2 Несмешённость скользящего экзамена

Теорема. Для оценки кроссвалидации имеет место

$$\mathbb{E}\check{R}^K(S_N, Q) = \mathbb{E}R(S_{N-\frac{N}{K}}, Q).$$

Как частный случай

$$\mathbb{E}\check{R}(S_N, Q) = \mathbb{E}R(S_{N-1}, Q).$$

Данный факт означает, что математическое ожидание оценки кроссвалидации равно среднему риску для решений, построенных тем же методом по выборкам размера $N - \frac{N}{K}$.

Этот факт мы будем называть несмешённостью скользящего экзамена.

При этом, делая утверждение о несмешённости оценки скользящего экзамена нужно обязательно делать оговорку, что несмешённость имеет место, если финальное решение строится по выборке размера $N - \frac{N}{K}$.

В реальной ситуации, когда финальное решение строится по выборке размера N , скользящий экзамен даёт вообще говоря смешённую оценку (несколько пессимистичную).

Доказательство теоремы элементарно, хотя и неочевидно.

Для начала докажем несмешенность оценки hold-out. Для простоты рассмотрим случай $K = N$, т.е. в каждом фолде один объект. Поскольку все фолды равноправны, возьмём первый фолд в качестве тестового. Требуется доказать, что

$$\mathbb{E}\check{R}_1(S, Q) = \mathbb{E}\mathcal{L}(y^1, \lambda_{Q, S'_1}(x^1)) = \mathbb{E}R(S_{N-1}, Q).$$

Данное тождество практически очевидно. Для доказательства, используются рассуждения, аналогичные рассуждениям в следующей задаче.

В урне находится 7 чёрных и 3 белых шаров. Наугад извлекли шар. Затем извлекли второй шар, который оказался белым. Какова вероятность, что и первый шар — белый?

Решение. Поскольку при извлечении первого шара мы не посмотрели на его цвет, ответ будет таким же, как если бы этот шар извлекался вторым, т.е. $\frac{2}{9}$.

Применительно к оценке hold-out можно аналогично сказать, что пока мы не использовали объекты из первого (отложенного) фолда, ожидаемая ошибка на них такая же, как на любых «новых» объектах.

Для завершения доказательства теоремы осталось использовать тот факт, что математическое ожидание суммы есть сумма математических ожиданий, причём для любых, в т.ч. для зависимых, случайных величин.

6.4.3 Некорректное оценивание дисперсии

Некоторые пользователи методов машинного обучения вычисляют среднеквадратичный разброс по фолдам.

Полученная величина не имеет никакого смысла. По крайней мере её категорически нельзя использовать как оценку дисперсии скользящего экзамена (т.е. как оценку его точности).

Вместо этого можно использовать другую оценку, которая тоже эвристическая, но она имеет хоть какие-то эмпирические обоснования.

6.4.4 Точность скользящего экзамена

Пусть критерий качества представляется математическим ожиданием функции потерь.

Обозначим $\xi_i = \mathcal{L}(y^i, \lambda_{k(i)}(x^i))$ – величину потерь на i -м объекте контрольной выборки.

Здесь λ_k – решающая функция, построенная по выборке с исключением k -го фолда, а $k(i)$ – номер фолда, которому принадлежит i -й элемент выборки.

Случайные величины ξ_i имеют одинаковое распределение. При этом они как правило зависимы. Под «как правило» здесь подразумевается, что зависимость имеет место во всех практически значимых ситуациях, хотя, вероятно, можно сконструировать искусственные примеры, когда зависимости нет.

В отличие от величин ζ_i , которые зависели только от одного объекта (контрольной) выборки, величины ξ_i являются функцией почти всей (обучающей) выборки (за исключением других объектов из того же фолда). При этом мы можем рассмотреть условную независимость величин ξ_i и ξ_j , когда все объекты выборки, кроме (x^i, y^i) и (x^j, y^j) , фиксированы. Величины, соответствующие объектам из одного фолда, будут условно независимы. Величины, соответствующие объектам, принадлежащих разным фолдам, вообще говоря, условно зависимы. Иными словами, величины ξ_i и ξ_j условно независимы, если $k(i) = k(j)$, и, как правило, зависимы, если $k(i) \neq k(j)$.

Может показаться, что зависимость между величинами ξ_i слабая и ей можно пренебречь. Это в корне неверно. Зависимость в некотором смысле действительно слабая, но пренебречь ей нельзя, и она существенно влияет на дисперсию оценки скользящего экзамена. Аналогично, можно заметить, что в эмпирическом риске и скользящем экзамене выборки, по которым ведётся обучение, различаются всего одним объектом. Но получившиеся оценки кардинально разные в плане смещения.

Математическое ожидание $\mu = E\xi_i$ есть средний риск решения, построенного по выборке объёма $N - K$.

Оценка скользящего экзамена в текущих обозначениях принимает вид

$$\bar{\mu} = \frac{1}{N} \sum_{i=1}^N \xi_i.$$

Мы можем рассматривать эту оценку как оценку риска для решения, построенного по всей выборке, так и как оценку среднего риска. В обоих случаях, чтобы оценить точность этой оценки, нужно оценить дисперсию $\sigma^2 = D\xi_i$.

Рассмотрим оценку

$$\tilde{\sigma}^2 = \frac{1}{N-1} \sum_{i=1}^N (\xi_i - \bar{\mu})^2.$$

Аналогичная оценка для контрольной оказалась несмешенной, однако при вычислении её математического ожидания мы сущ-

ственno использовали независимость (достаточно некоррелированности) величин ζ_i .

Поскольку величины ξ_i зависимы, вычислить математическое ожидание $\tilde{\sigma}^2$ не получится, и в общем случае оно не равно σ^2 .

Но даже если бы мы получили несмещённую оценку для σ^2 , это не дало бы нам оценку для $D\bar{\mu}$, поскольку в общем случае дисперсия суммы не равна сумме дисперсий. И тогда $D\bar{\mu} \neq \frac{1}{N}\sigma^2$.

Для скользящего экзамена в общем случае неизвестен доверительный интервал и нет хороших оценок дисперсии.

Эмпирические исследования показывают, что можно использовать следующую эвристическую оценку

$$D\bar{\mu} \approx \frac{C}{N}\tilde{\sigma}^2, \quad (6.4)$$

где $C \geq 1$ – некоторый поправочный коэффициент, значение которого зависит от метода и задачи. На практике бывает несколько больше 1, но не превосходит 2.

Пусть J_k – множество индексов объектов, принадлежащих k -му фолду, а N_k – размер фолда.

Обозначим

$$\bar{\mu}_k = \frac{1}{N_k} \sum_{i \in J_k} \zeta_i.$$

Для удобства будем считать, что N делится на K . Положим все N_k одинаковыми, т.е. $N_k \equiv \frac{N}{K}$.

Тогда все μ_k будут вообще говоря зависимыми одинаково распределёнными случайными величинами со средним μ и дисперсией σ_K^2 , которая неизвестна как связана с σ^2 .

Оценка

$$\tilde{\sigma}_K^2 = \frac{1}{K-1} \sum_{k=1}^K (\bar{\mu}_k - \bar{\mu})^2$$

будет вообще говоря смещённой оценкой для σ_K^2 .

В некоторых работах в качестве оценки для $D\bar{\mu}$ используют величину $\frac{\tilde{\sigma}_K^2}{K}$.

Это по сути грубая ошибка.

В разделе о контрольной выборке мы выяснили, что такая оценка при $K < N$ имеет большую погрешность и её использование не имеет рационального оправдания. Использовать её, например, при $K = 10$ – это практически то же самое, что взять «с потолка» какие-то произвольные числа.

При $K = N$ получаем $\check{\sigma}_K^2 = \tilde{\sigma}^2$. Но в этом случае нужно использовать поправку, учитывающую, что дисперсия скользящего экзамена существенно выше, чем у контрольной выборки.

Можно ли по скользящему экзамену построить хоть какую-то несмешённую оценку дисперсии? Отчасти, да. Оценка

$$\check{\sigma}_K^2 = \frac{1}{N - K} \sum_{j=1}^N (\xi_j - \bar{\mu}_{k(j)})^2$$

является несмешённой оценкой для средней дисперсии потерь для решения, построенного по выборке объёма $N - K$. Однако эта величина также не поможет нам оценить $D\bar{\mu}$.

Сочетание «средняя дисперсия» может вызвать недоумение, поскольку дисперсия сама по себе подразумевает усреднение. Под средней дисперсией здесь понимается величина $E D_{\lambda_k(i)} \xi_i$, т.е. дисперсия берётся при фиксированной решающей функции.

Это понятие аналогично среднему риску. Заметим, что средняя дисперсия — это не то же, что σ^2 .

6.4.5 Разновидности скользящего экзамена

Идея скользящего экзамена допускает множество реализаций и вариантов. Перечислим некоторые из них:

- leave-one-out,
- k -fold crossvalidation,
- случайные подвыборки,
- повторные разбиения,
- кроссвалидация со стратификацией,

- bootstrap.

Про первые варианты мы уже рассказали, кратко опишем остальные.

Идея случайных подвыборок заключается в том, что выборка разделяется на тестовую и обучающую части путём случайного выбора (без повторения) объектов для теста. При этом долю объектов для теста выбирают не слишком большой (до 20%), чтобы не было заметного смещения. На оставшихся объектах проводится обучение и для выбранных делается прогноз. Данные шаги проделываются многократно. Для каждого прогноза вычисляется функция потерь, и все потери усредняются.

Данный метод имеет свойства, схожие с методом кроссвалидации, однако, требует больших вычислительных затрат (нужно проводить итерации, пока почти все объекты не побывают в роли тестовых), по-видимому, не давая заметного повышения точности. Более предпочтительной выглядит кроссвалидация с повторением.

Идея повторных разбиений заключается в том, процедура кроссвалидации проводится многократно, причём на каждой итерации выборка случайным образом «перемешивается». В качестве оценки риска берётся среднее значение потерь по всем итерациям.

Повторные разбиения обычно позволяют несколько повысить точность оценки риска, но эффект, как правило, незначителен. С другой стороны, вреда от такой процедуры быть не может, поэтому провести несколько итераций бывает оправдано.

Встречается грубое заблуждение, будто большим числом повторений можно сколь угодно уменьшать дисперсию оценки. На самом деле, все повторы делаются на одной и той же выборке, поэтому получаемые оценки сильно зависимы и дисперсия оценки практически не уменьшается. Иными словами, чудес не бывает, и никому ещё не удалось успешно применить метод Мюнхгаузена по вытаскиванию себя из трясины.

Также встречается заблуждение, что точность оценки можно существенно повысить, увеличивая число фолдов. Оно основано видимо на том, что при независимых измерениях ошибка уменьшается как $\frac{1}{N}$, поэтому ожидают, что при K фолдах ошибка будет

порядка $\frac{1}{K}$. На самом деле, дисперсия оценки если и зависит от K , то несущественно.

Здесь необходимо сделать важную оговорку. В некоторых случаях увеличение K или же применение повторных разбиений всё же дают заметный эффект.

Он объясняется следующим фактором. Многие (пожалуй даже большинство) методов имеют не вполне детерминированное поведение. Многие в явном виде используют случайность (которой можно управлять через `random_seed`). У многих получаемое решение зависит от порядка следования объектов выборки.

В результате, по одной и той же выборке мы можем получить одним методом разные решения. Понятно, что такой разброс даёт «лишнюю прибавку» к погрешности (дисперсии) оценок. И именно эта добавка уменьшается за счёт повторных разбиений.

Но такой же эффект мы можем получить, «усовершенствовав» сам метод, а именно включив в него усреднение решений с разным `random_seed` и с разным порядком объектов в обучении.

Если в задаче классификации классы несбалансированы, т.е. вероятности классов сильно отличаются, то при разбиении выборки в разных фолдах частота объектов «редкого» класса может сильно различаться (например, в какой-то фолд могут вообще не попасть объекты одного из классов).

Идея стратификации заключается в том, чтобы разбивать выборку так, чтобы соотношение классов по фолдам было примерно одинаковым. Этого легко добиться, например, отдельно разбить на фолды объекты каждого класса, а затем соответствующие фолды объединить.

Сложно сказать, стоит ли ожидать от стратификации значительного эффекта. Обучение при кроссвалидации проводится на выборках, мало отличающихся от исходной, т.е. баланс классов в них сохранён, и кажется, что стратификация не должна заметно влиять на оценку.

Наиболее полезные выводы относительно разных вариантов кроссвалидации можно сделать на основе статистического моделирования. Пример такого моделирования приведён в конце данной главы.

Автор рекомендует читателям провести аналогичные эксперименты и сделать собственные выводы относительно лучшего способа оценки риска.

Оценка bootstrap есть

$$\check{R}(S, Q) = \mathbb{E} \frac{1}{|J_0|} \sum_{i \in J_0} \mathcal{L}(y^i, \lambda_{Q, \check{V}}(x^i)),$$

где \check{V} – выборка, получаемая из S путем N -кратного случайного (равновероятного) выбора ее значений с повторениями, J_0 – множество индексов объектов из S , ни разу не выбранных в \check{V} , математическое ожидание \mathbb{E} подразумевает усреднение по бутстрэп выборкам \check{V} .

Ввиду того, что оценка bootstrap является смещенной (поскольку для обучения используется лишь $1 - e^{-1}$ доля объектов), чаще используют ее в комбинации с эмпирическим риском

$$\ddot{R}(S, Q) = e^{-1} \check{R}(S, Q) + (1 - e^{-1}) \check{R}(S, Q).$$

Автору неизвестны убедительные свидетельства о преимуществах оценки bootstrap по сравнению с типичной кроссвалидацией.

6.4.6 Оценка out-of-bag

Чаще всего скользящий экзамен используется в виде K -fold кроссвалидации, когда обучающая выборка разбивается на K непересекающихся подмножеств (фолдов) и при обучении тестируемый объект удаляется из выборки вместе со всем фолдом, которому он принадлежит.

Как правило, после проведения скользящего экзамена проводится обучение метода уже на всей выборке, и полученная решающая функция используется в качестве итогового решения.

Однако в качестве итогового решения можно использовать решающие функции, уже построенные в ходе кроссвалидации, просто объединив их в ансамбль.

В обоих случаях оценкой качества выступает одно и то же выражение – функционал скользящего экзамена. Однако решающие функции, качество которых он оценивает, разные. Поэтому

с формальной точки зрения это разные оценки. И во втором случае будем называть этот функционал оценкой out-of-bag.

Таким образом, сравнение оценок out-of-bag и скользящего экзамена фактически означает сравнение двух способов построения итогового решения (а функционал качества в обоих случаях одинаковый).

Заметим, что первоначально оценка out-of-bag появилась в методе бэггинга (RandomForest), где она выглядит несколько иначе, чем мы определили, однако основная идея та же.

Оценка out-of-bag является, вообще говоря, смешённой.

Например, в методе RandomForest для каждого объекта выборки число деревьев, которые не использовали этот объект для обучения, составляет примерно треть, т.е. $\frac{1}{e}$, от общего их количества. Получается, что оценка out-of-bag соответствует качеству леса, примерно втрое меньшего размера, поэтому такая оценка риска несколько завышена.

В целом, использование out-of-bag представляется удачным сочетанием идей скользящего экзамена и ансамблирования.

6.4.7 Свойства оценки скользящего экзамена

Мы установили следующие свойства скользящего экзамена:

- относительная простота вычисления,
- несмешённость (не в том смысле, в котором хотелось бы),
- состоятельность (может быть не доказана),
- большая дисперсия, для которой нет хороших оценок,
- среднеквадратичный разброс по фолдам (практически) никак не связан с дисперсией скользящего экзамена и не может выступать в роли её оценки..

Несмешённость данной оценки не означает, что получив некоторое значение на кроссвалидации, мы вправе оценить риск этим значением. Как и в случае контрольной выборки, его следует скорректировать в большую сторону.

Что касается состоятельности, то для кроссвалидации с фиксированным K она очевидна (следует из состоятельности оценки hold-out), но при $K = N$ простого доказательства не получается (в виду зависимости получаемых величин).

Любой вариант скользящего экзамена имеет большую дисперсию. Есть эмпирические свидетельства, что дисперсия скользящего экзамена бывает до двух раз выше дисперсии оценки по контрольной выборке.

У скользящего экзамена есть множество вариантов, однако опыт свидетельствует, что они примерно равнозначны по качеству, а значит имеет смысл использовать что проще.

Если вычислительные ресурсы ограничены, то разумный выбор — кроссвалидация с $K = 5$. Если ресурсов с избытком, то имеет смысл взять $K = 10$. Предварительно объекты следует «перемешать», поскольку в исходной таблицы они могли идти в неслучайном порядке.

6.5 Доверительный интервал для риска

6.5.1 Качество точечных оценок

Вернёмся к вопросу о том, какая из оценок риска лучше (точнее). Для начала нужно понять, что значит, одна оценка точнее другой и как эту точность определять.

В общем случае оценочный функционал — это некоторая функция выборки.

Качество оценок обычно характеризуют средним квадратом уклонения. Для эмпирического функционала $\bar{R}(V, Q)$ можем записать

$$\Delta = E(\bar{R}(S, Q) - R(c, \lambda_{Q,S}))^2.$$

Существенная проблема заключается в том, что выражения зависят от c — распределения, которое неизвестно.

Чтобы получить гарантированную оценку точности, можно взять максимум Δ по всем c .

Но «цена погрешности» зачастую бывает разной для разных значений риска. Например, для вероятности ошибки если мы име-

ем оценку 0,6 при истинном значении 0,5 — это вполне несущественное отличие. Но если мы оценили вероятность ошибки как 0,1, а на самом деле она равна 0,2 — такое отклонение очень значительно. Иными словами, одно и то же отклонение при разных значениях риска имеет разную значимость.

Кажется, что нужно оценивать погрешность оценки при условии полученного значения этой оценки, например использовать

$$\mathbb{E}_{\bar{R}} R(c, \lambda_{Q,S}).$$

Смысл данного выражения — это например какая ожидаемая вероятность ошибки будет у решения, при условии, что на кроссвалидации получили ноль ошибок.

В случае контролькой выборки мы сумели вычислить подобное выражение, но использовали байесовский подход (который включает необоснованный произвол), и на другие функционалы этот подход не переносится.

Если же взять максимум по всем распределениям, то результат получится вырожденным. В частности, вероятность ошибки будет 0,5, независимо от значения функционала в условии.

Дело в том, что при совпадающих распределениях двух классов вероятность ошибки будет 0,5, но при этом вероятность получить ноль ошибок на кроссвалидации будет ненулевой. То что она при этом ничтожно малая, никак не влияет на возможность вычислить условное математическое ожидание риска, которое будет, очевидно, 0,5.

Здесь естественно применить идею проверки гипотез, а именно исключить (при взятии максимума) те распределения, при которых вероятность получить наблюдаемый результат слишком мала.

В этом варианте конструкция получается осмысленной, но слишком сложной. Раз уж мы поняли неизбежность использования идей проверки гипотез, естественно строить доверительные интервалы.

6.5.2 Доверительный интервал для риска

Доверительный интервал для R зададим в виде $[0, \hat{R}_\eta(S)]$. Здесь мы ограничиваемся односторонними оценками, поскольку на практике для риска важны именно оценки сверху. Таким образом, в данном случае построение доверительного интервала эквивалентно выбору функции $\hat{R}_\eta(S)$, которую назовём оценочной функцией.

При этом должно выполняться условие:

$$\forall c, P_c(R \leq \hat{R}(S)) \geq \eta, \quad (6.5)$$

где η – заданная доверительная вероятность.

Может показаться, что функцию $\hat{R}(S)$ правильнее называть оценкой доверительного интервала, поскольку в классическом определении требуется, чтобы вероятность попадания оцениваемой величины в интервал должна быть равна доверительной вероятности η , а у нас в условии стоит «больше или равно». Однако в данном случае обеспечить для всех распределений равенство невозможно, поэтому доверительного интервала в таком классическом понимании не существует. Говорить об оценке того, что не существует, очевидно некорректно. Поэтому будем называть $\hat{R}(S)$ доверительным интервалом, помня о различии с классическим определением.

На практике интервальную оценку будем строить как $\hat{R}(\bar{R}(S))$ – функцию точечной оценки.

Условие 6.5 не определяет доверительный интервал однозначно, и остаётся произвол в выборе конкретного вида оценочной функции, но эта проблема решаема. Предположим, что мы выбрали какую-то функцию $\hat{R}(\bar{R}(S))$ эвристически. Вопрос, как проверить выполнение условия 6.5, которое требует перебрать все возможные распределения P_c .

Пожалуй, самый реалистичный вариант — это опять же эвристически выбрать некоторое множество семейств распределений. Так мы приходим к идее эмпирических доверительных интервалов для риска. Идея заключается в том, что если некоторое условие выполняется при всех распределениях, которые мы только смогли выдумать, то можем надеяться, что оно окажется выпол-

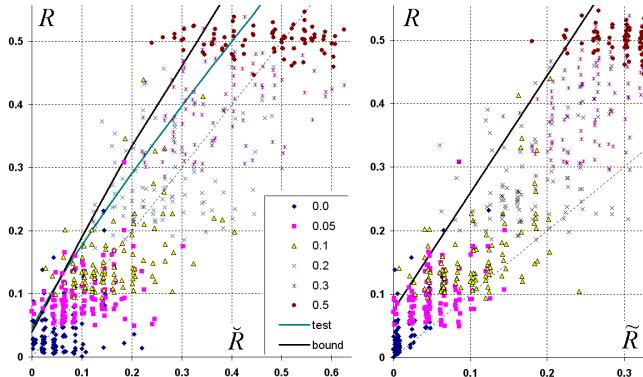


Рис. 6.4: Эмпирические доверительные интервалы на основе скользящего экзамена и эмпирического риска.

ненным и для реального распределения, с которым мы столкнёмся в реальной задаче.

Эмпирический доверительный интервал для R зададим в виде $[0, \hat{R}(\bar{R}(S))]$.

При этом должно выполняться условие:

$$\forall c \in \tilde{C}, P_c(R \leq \hat{R}(\bar{R}(S))) \geq \eta,$$

где η – заданная доверительная вероятность, а \tilde{C} – эвристически выбранное множество распределений.

На рис. 6.4 приведена иллюстрация идеи построения эмпирических доверительных интервалов.

Каждая точка на графике соответствует некоторой случайной выборке, по которой построена решающая функция. Для этой решающей функции вычисляется риск и его оценка через скользящий экзамен (левая диаграмма) или эмпирический риск (правая диаграмма). Разные цвета точек (виды маркеров) соответствуют различным распределениям.

Кривые, помеченные как «bound», проведены таким образом, чтобы выше них оказывалось бы не более 10% точек из каждого распределения. Они могут выступать в роли оценочной функции \hat{R} .

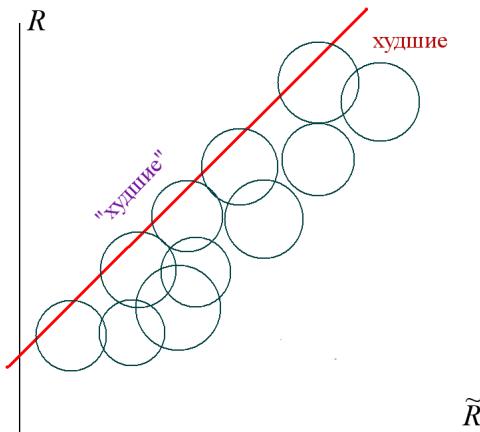


Рис. 6.5: Иллюстрация понятия «худшее распределение».

Поскольку оценки типа 6.5 подразумевают справедливость при любых распределениях, фактически они ориентированы «на худший случай». Часто это отмечают как недостаток рассмотренного подхода.

Действительно, известно много примеров, когда оценки для худшего случая оказывались далеки от типичных значений в реальных ситуациях. Например, симплекс-метод в худшем случае имеет экспоненциальную трудоёмкость, но в реальных задачах, как правило, находит решение быстро.

Рисунок 6.5 иллюстрирует, что распределение может быть «плохим» в разных смыслах. Чаще всего плохим называют распределение, при котором любой метод даёт решение низкого качества (с большим значением риска). Это например распределение, в котором распределения переменных при условии каждого класса совпадают. Подобные распределения в реальных задачах действительно редко встречаются, поэтому ориентироваться на них при конструировании оценок — неразумно.

Но оценка 6.5 ориентирована на распределения, которые плохие с совсем другом смысле. Это распределения, для которых расхождение эмпирической оценки риска и риска максимальны. Но значения самого риска при этом могут быть небольшими.

Подобные «плохие» распределения уже не являются чем-то экзотическим и вполне типичны для реальных данных.

В частных случаях удалось даже явно найти эти «худшие» распределения, и оказалось, что они обладают такой особенностью, что в некоторой подобласти пространства переменных классы перекрываются, а в других областях — хорошо разделимы.

Эмпирические доверительные интервалы можно строить не только по одной статистике, но и по нескольким, например, одновременно используя эмпирический риск и скользящий экзамен.

6.6 Выводы

По материалам главы можно сделать следующие выводы:

- наилучший способ оценивания риска неизвестен,
- на практике обычно используют скользящий контроль,
- все разновидности скользящего контроля примерно равнозначны,
- нет точных оценок доверительного интервала для риска по обучающей выборке,
- полезно использовать статистическое моделирование.

Оценивание качества решений — одна из самых важных и сложных проблем машинного обучения, которая до настоящего времени остаётся открытой.

6.7 Примеры кода

Приведём пример кода для моделирования построения решений на случайных выборках.

Автор настоятельно рекомендует читателям провести подобный эксперимент, чтобы непосредственно понаблюдать поведение оценок скользящего экзамена.

Сначала рассмотрим функцию генерации данных.

Accuracy

	train	true	test	cv5	cv10	cv5_rep	cv10_str
mean	0.9335	0.8429	0.8423	0.8379	0.8422	0.838	0.8386
std	0.0091	0.0068	0.0122	0.0121	0.0129	0.012	0.0129

Logloss

	train	true	test	cv5	cv10	cv5_rep	cv10_str
mean	0.2641	0.4047	0.3984	0.4045	0.4017	0.4047	0.4033
std	0.0201	0.0078	0.0208	0.0229	0.0230	0.0219	0.0219

Рис. 6.6: Результаты моделирования. В верхней таблице значения точности, в нижней — логарифмические потери.

Аргументами функции являются размер выборки, число переменных и байесовский уровень ошибки. Все переменные, за исключением первых двух, являются шумовыми (не зависят от целевой переменной).

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import KFold,
                                    StratifiedKFold, RepeatedKFold

def gen_data(N,n,by_err):
    df = pd.DataFrame(np.random.uniform(-1.0, 1.0, (N,n)),
                      columns=[f'X{i}i' for i in range(n)])
    A = (df['X1'] * df['X1'] - df['X1'] * df['X2'] +
         df['X2'] * df['X2'] < 0.7**2).astype(int)
    df['Y'] = np.random.uniform(0.0, 1.0, N)
    df['Y'] = (df['Y'] >= (by_err + (1 - 2 * by_err) * A)).astype(int)
    return df

df = gen_data(1000,3,0.1)
ax = df.loc[df['Y']==0].plot.scatter(x='X1',y='X2',
                                      figsize=(8,8))
df.loc[df['Y']==1].plot.scatter(x='X1',y='X2',c='red', ax=ax)
```

В конце приведённого фрагмента сгенерированная выборка

визуализируется в пространстве информативных переменных, и можно наглядно посмотреть форму распределения.

Ниже приведены функция вычисления оценки точности по кроссвалидации и функция моделирования.

```
def crossval_score(X,y,model,kf):
    scor = []
    for train , test in kf.split(X,y):
        X_train , X_test = X.iloc[train] , X.iloc[test]
        y_train , y_test = y.iloc[train] , y.iloc[test]
        clf = model.fit(X_train,y_train)
        scor.append( clf.score(X_test,y_test) )
    return np.array(scor).mean()

def simulate():
    model = GradientBoostingClassifier(n_estimators=150,
                                         learning_rate=0.025, max_depth=5,
                                         random_state=0)
    df = gen_data(1000,5,0.1)
    df1 = gen_data(1000,5,0.1)
    dft = gen_data(100000,5,0.1)
    X = df.iloc[:, :-1]
    y = df.iloc[:, -1]
    model.fit(X,y)
    scores = {
        'train' : model.score(X,y),
        'true' : model.score(dft.iloc[:, :-1],dft.iloc[:, -1]),
        'test' : model.score(df1.iloc[:, :-1],df1.iloc[:, -1]),
        'cv5' : crossval_score(X,y,model,KFold(5)),
        'cv10' : crossval_score(X,y,model,KFold(10)),
        'cv5_rep' : crossval_score(X,y,model,
                                    RepeatedKFold(n_splits=5,n_repeats=3)),
        'cv10_str' : crossval_score(X,y,model,
                                    StratifiedKFold(n_splits=5)) }
    return scores

dr100 = pd.DataFrame([simulate() for i in range(100)])
dr100.agg(['mean','std']).round(4)
```

Функция `simulate` генерирует случайную выборку и на ней проводит обучение модели, а также вычисление различных оценок качества, в том числе методом кроссвалидации. Функция вызывается 100 раз, и полученные результаты помещаются в таблицу `dr100`.

Результат выполнения кода приведён в верхней таблице на рис. 6.6.

Столбец `train` содержит эмпирический риск, `true` — истинное значение риска (оценено по выборке очень большого размера, и его можно считать точным), `test` — оценка риска по дополнительной контрольной выборке того же размера, `cv5` — оценка методом кроссвалидации при $K = 5$, `cv10` — оценка методом кроссвалидации при $K = 10$, `cv5_grep` — оценка методом кроссвалидации при $K = 5$ с тремя повторениями, `cv10_str` — оценка методом кроссвалидации при $K = 10$ со стратификацией.

Строка `mean` содержит средние значения величин. Видим, что смещение всех оценок (кроме `train`) находится в пределах статистической погрешности, т.е. несущественны.

Строка `std` содержит среднеквадратические (стандартные) отклонения величин. Видим, что между разными вариантами кроссвалидации нет существенных отличий.

Несколько нетипичен факт, что дисперсия кроссвалидации оказалась не выше, чем у контрольной выборки. Как правило, она оказывается всё же выше (что показывает уже следующий пример). Но даже при равенстве дисперсий, оценка по контрольной выборке будет точнее, т.к. погрешность кроссвалидации включает также и дисперсию величины `true`.

Поскольку ожидается, что стратификация имеет больший эффект при несбалансированных классах, следующий пример использует несколько изменённую модель (уменьшен размер эллипса), в которой частоты классов соотносятся как 1 к 10. Поскольку точность в таких задачах вычислять бессмысленно, в качестве критерия использована логарифмическая функция потерь.

Результат приведён в нижней таблице рисунка. Как видим, стратификация опять не даёт эффекта.

По результатам моделирования можно вычислить также корреляцию между величинами. Полученные значения трудно интерпретировать, однако они могут представлять интерес. В некоторых примерах корреляция между `true` и `cv5` оказывается даже отрицательной.

Подобные эксперименты, разумеется, не доказывают равно-

ценность разных вариантов кроссвалидации. Зато они потенциально могут продемонстрировать их неравноценность. Автору не удалось встретить или построить примеров, когда какой-то вариант кроссвалидации (при $K \geq 5$) был бы существенно лучше других. Возможно, такие примеры удастся найти читателям.

Глава 7

Задача машинного обучения

В данной главе мы, наконец, попытаемся выяснить, чем же до сих пор занимались, а именно, как формально поставить задачу, которую решали. В этом мы следуем логике движения от частного к общему. До этого не было смысла делать общую формальную постановку задачи, поскольку для её понимания нужно было получить содержательное представление о задаче.

Но перед общей постановкой требуется рассмотреть ещё ряд вопросов.

7.1 Критерии качества

7.1.1 Целевая функция потерь

Под качеством моделей машинного обучения в первую очередь понимают точность прогноза, которую они обеспечивают. Однако точность можно определять разными выражениями, поэтому в каждой задаче стоит выбор подходящей меры качества (точности).

Можно ли ввести объективную меру качества? Кажется, что да.

Заметим, что прогноз нам нужен в конечном итоге для принятия

тия каких-то решений. Например, на основе прогноза погоды принимаются решения, брать ли на улицу зонт, когда сеять и т.п. И каждое решение имеет свою объективную цену в случае, если оно окажется ошибочным. В большинстве случаев эту цену ошибки очень трудно точно измерить. Например, поставлен ошибочный диагноз. Пострадавший может подать иск о денежной компенсации, и в случае его удовлетворения это будут реальные денежные потери того, кто допустил ошибку. Но отражает ли эта компенсация реальный ущерб? И как быть, если из-за этой ошибки подавать иск уже некому?

В подавляющем большинстве случаев с приемлемой точностью оценить ущерб от ошибочного прогноза не представляется возможным. Но нужно понимать, что этот ущерб объективно существует.

Иными словами, можно считать, что во всех задачах машинного обучения существует объективная функция потерь, которую мы не знаем. Тем не менее качественное представление о величине потерь при разных исходах обычно имеется. И именно эти представления и нужно использовать при выборе функции потерь, т.е. выбирать выражение, которое хотя бы примерно соотносится с ожидаемым ущербом.

Виды критериев качества.

- Целевой критерий: функция потерь отражает реальные потери от ошибочно принятого решения. Мы не выбираем этот критерий, он — объективная данность.
- Критерий для обучения: может использовать значение целевой функции потерь на выборке, а может включать любую другую функцию потерь. Не обязательно вообще выражаться через потери (дискриминант Фишера). Обычно включает регуляризацию.

Целевой критерий качества в инженерной среде принято называть метрикой качества. Это, конечно, неудачный термин, поскольку метрика в математике — это расстояние, т.е. имеет совершенно другой смысл.

На самом деле, «критерий качества» — тоже неудачный термин, поскольку в математике термин «критерий» так или иначе подразумевает предикат, т.е. логическую функцию, а не числовую.

За неимением лучшего, будем использовать данные термины, и под целевым критерием (метрикой) качества будем понимать математическое описание целевой функции потерь.

Критерий для обучения не обязательно совпадает с целевым. Зачастую методы машинного обучения умеют оптимизировать лишь небольшое число функций, поэтому выбор критериев для обучения довольно узок.

7.1.2 Критерии точности классификации

Матрица ошибок. Ошибки первого и второго рода.

True Positive (TP)	False Positive (FP)
False Negative (FN)	True Negative (TN)

Из этих частот получаем: accuracy, precision, recall, specificity.

Заметим, что на русский язык термины accuracy и precision переводятся одинаково, поэтому под точностью будем понимать accuracy, а precision использовать без перевода.

Также можно построить ROC-кривую и PR-кривую.

Кривая ошибок

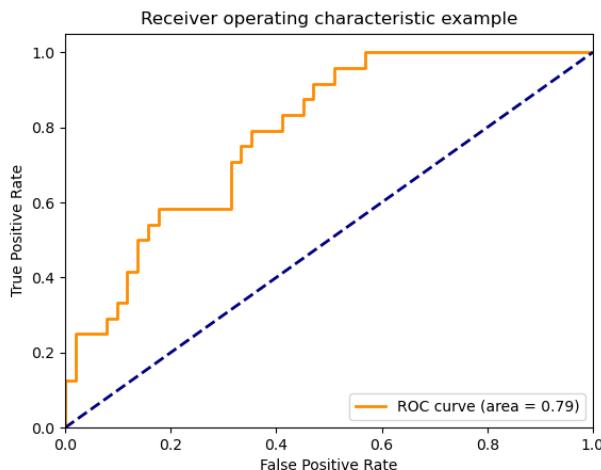
Мы всегда можем повысить точность прогноза одного класса за счёт увеличения числа ошибок на другом классе, поэтому более полезной информацией будет именно зависимость между данными величинами, а не их частные значения.

Распространённым критерием качества оценки $\tilde{g}(x)$ является AUC — area under the curve, т.е. площадь под так называемой ROC-кривой (receiver operating characteristic, или кривая ошибок).

Пусть $F_{\tilde{g}}^y(z)$ — условная функция распределения случайной величины $\tilde{g}(x)$, определяемая условной мерой $P_c(E \mid y)$, где $E \subseteq X$ — событие.

Тогда ROC-кривая определяется как кривая, заданная параметрически множеством точек $(F_{\tilde{g}}^{-1}(z), F_{\tilde{g}}^1(z))$, когда z изменяется от $-\infty$ до $+\infty$, и отрезков, соединяющих последовательные точки в случае разрывов функций распределения.

Пример кривой ошибок



Свойства кривой ошибок

Начало ROC-кривой в точке $(0, 0)$, конец — в $(1, 1)$.

Чем больше значение AUC, тем лучше решение $\tilde{g}(x)$.

Значение 0,5 соответствует наихудшему качеству $\tilde{g}(x)$, учитывая что $AUC(1 - \tilde{g}(x)) = 1 - AUC(\tilde{g}(x))$.

Строго монотонное преобразование функции $\tilde{g}(x)$ не меняет AUC.

Для «обычной» решающей функции значение AUC есть просто среднее арифметическое между вероятностями правильного прогнозирования каждого класса.

На выборке используются эмпирические функции распределения.

7.1.3 Функция потерь для оценки вероятности

Возможна более общая постановка задачи, когда под решающей функцией понимается оценка $\tilde{g}(x)$ условной вероятности

$$g(x) = \mathsf{P}_c(y = 1 \mid x) = \frac{\mathsf{P}_c(dx, y = 1)}{\mathsf{P}_c(dx)}.$$

Качество решения $\tilde{g}(x)$ можно выражать следующей функцией потерь

$$\mathcal{L}(y, \tilde{g}(x)) = -I(y = 1) \cdot \ln \tilde{g}(x) - I(y = -1) \cdot \ln(1 - \tilde{g}(x))$$

Выборочное среднее данной функции потерь есть взятая со знаком минус функция правдоподобия выборки по отношению к оценке условной вероятности.

Критерии качества классификации

- Точность (accuracy), balanced accuracy.
- F1 score.
- ROC-AUC, Gini coefficient.
- Logloss.

7.1.4 Функция потерь для непрерывной целевой переменной

Обычно используют среднеквадратичное или среднее абсолютное отклонение.

В первом случае оптимальный прогноз — функция регрессии, во втором — условная медиана.

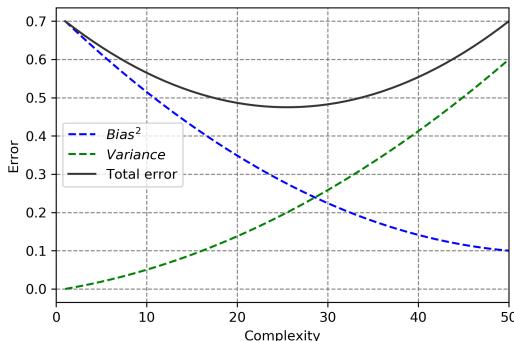
7.2 Разложение ошибки

Разложение критерия качества решающих функций Есть (как минимум) два подхода к разложению критерия качества решающих функций.

- Разложение на смещение и разброс (bias-variance decomposition).
- Разложение на меру адекватности и меру статистической устойчивости [6]

7.2.1 Bias-variance decomposition

Желаемое поведение разложения



- смещение \neq ошибка аппроксимации
- разброс \neq ошибка статистического оценивания

Смещение и разброс

Для независимых случайных величин u и v имеет место

$$\mathbb{E}(u - v)^2 = \mathbb{D}u + (\mathbb{E}u - \mathbb{E}v)^2 + \mathbb{D}v$$

Зафиксируем точку x признакового пространства и подставим $u = y | x$, $v = f(x)$.

Усреднение ведётся по условному распределению на целевой переменной и по выборкам заданного объёма.

Получаем «шум», смещение и разброс (в точке x).

Логарифмическая функция потерь Логарифмическая функция потерь

$$L(y, u) = -\ln u_y$$

Критерий эквивалентен функции правдоподобия выборки.

Можно свести к дивергенции Кульбака-Лейблера.

$$L(v, u) = \sum_{\omega=1}^k v_\omega \ln \frac{v_\omega}{u_\omega}.$$

Универсальное разложение

Имеет место

$$\mathbb{E}L(y, u) = H(p) + L(p, \bar{u}) + \mathbb{E}L(\bar{u}, u)$$

Здесь \bar{u} – «главное предсказание», но на самом деле это распределение, при котором прогноз оптимален.

Разложение для kNN

В ряде источников (напр., [2]) приводится следующая формула разложения для метода kNN

$$\mathbb{E}(y - f(x))^2 | x = \left(f(x) - \frac{1}{k} \sum_{i=1}^k f(N_i(x)) \right)^2 + \frac{\sigma^2}{k} + \sigma^2.$$

В классической постановке задачи регрессии, т.е. когда координаты X в обучающей выборке фиксированы (не случайны), первое слагаемое есть bias.

В общей постановке, когда X случайны, первое слагаемое содержит как смещение, так и разброс.

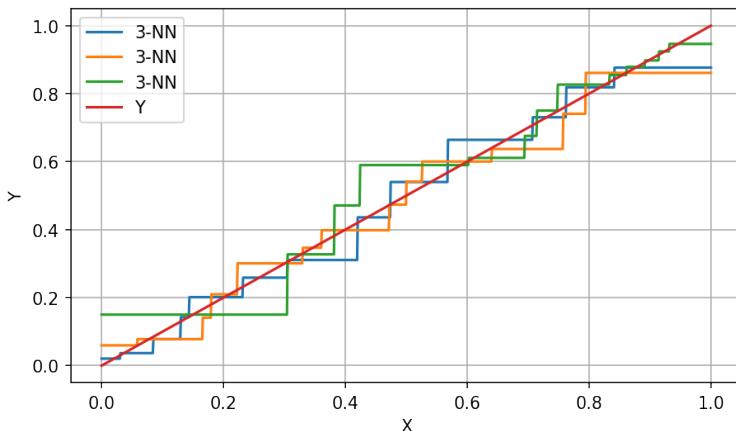
kNN для одномерной регрессии

Смещение существенно только на краях области.

Разложение для kNN

Без учёта краевых эффектов имеем разложение

$$\mathbb{E}(y(x) - f(x))^2 \approx \frac{(k+1)(k+2)}{12N^2k} + \frac{\sigma^2}{k} + \sigma^2. \quad (7.1)$$



Первое и второе слагаемые — оба variance.

Как видим, с ростом сложности variance сначала уменьшается и только потом растёт.

При большей размерности пространства эффект исчезает.

7.2.2 Теория статистической устойчивости

Меры адекватности и устойчивости

Идея подхода [6] в том, чтобы разложить ошибку на погрешность аппроксимации и статистическую погрешность.

Асимптотическое значение среднего качества (риска)

$$R = \lim_{N \rightarrow \infty} \mathbb{E} L(y, u)$$

Мера адекватности есть разность между асимптотическим средним риском и байесовским уровнем ошибки.

Мера статистической устойчивости есть разность между средним риском и асимптотическим.

7.2.3 Процесс обучения

Отступ как мера сложности

В качестве универсальной меры сложности решения можно использовать отступ (ненормированный).

Нормированный отступ фигурирует в оценках обобщающей способности.

Неномированный отступ характеризует степень обучения (и переобучения).

Отступ вычисляется на основе оценок вероятности.

Процесс обучения

- Ошибка на обучении монотонно уменьшается (если это не так, значит обучение «пошло вспять»).
- Иногда можно обучаться и после достижения нулевой ошибки.
- По кроссвалидации можно оптимизировать предельное значение сложности, а в рамках порога выбирать минимум на обучении.

7.2.4 Выводы

Выводы Результаты исследования (проведённые численно и аналитически) показывают, что в ряде примеров как смещение, так и разброс, могут вести себя немонотонно при увеличении сложности (сначала уменьшаться, а затем расти).

В следствие этого разложение на смещение и разброс не является удовлетворительным объяснением процесса обучения, поскольку смещение и разброс лишь косвенно характеризуют аппроксимационную способность и статистическую устойчивость метода построения решающих функций.

В качестве универсальной меры сложности решения можно использовать отступ (ненормированный).

7.3 Общая постановка задачи машинного обучения

7.3.1 Модели

В машинном обучении очень широко используется термин «модель».

Обычно он используется как синоним понятию решающая функция, либо под моделью понимают класс решающих функций.

Мы также будем использовать это значение термина, но также будем говорить о вероятностных моделях.

Под вероятностной моделью будем понимать по сути просто распределение, из которого получены данные как случайная независимая выборка.

Такая вероятностная модель является полной описательной моделью. Она позволяет смоделировать задачу, т.е. сгенерировать любой объём данных. Также по полной вероятностной вероятностной модели всегда можно принять оптимальное решение (минимизирующее ожидаемые потери).

Есть описательные модели, а есть модели принятия решения. Вторые всегда неполные.

Фактически, если известны распределения, то нет никакой проблемы машинного обучения, есть разве что чисто техническая проблема превращения полного знания в оптимальное решение (например, путём использования формулы Байеса для нахождения условной вероятности класса в заданной точке). Тем не менее мы можем тестировать прогностические модели, «обучая» их на распределениях, оценивая тем самым их аппроксимирующую способность.

По полной описательной модели (вероятностной) можно принять оптимальное решение. Но можно строить решение и без описательной модели. Достаточно наличия объективной функции потерь и обучающей выборки.

Часто задачу машинного обучения формулируют как задачу поиска скрытой зависимости.

Но объективно никакой зависимости нет, есть только совместное распределение.

Задача же определяется критерием качества, например, для квадратичной функции потерь оптимальным решением будет регрессия, а для абсолютных потерь — условная медиана.

Варианты:

- регрессионный анализ — не всегда строим регрессию;
- восстановление зависимостей — объективной зависимости обычно нет;
- построение решающих функций — самый точный термин.

7.3.2 Задача построения решающей функции

Основные понятия Пусть X — пространство значений переменных, используемых для прогноза,

$Y = \{-1, 1\}$ — пространство значений прогнозируемых переменных,

C — множество всех вероятностных мер на заданной σ -алгебре подмножеств множества $D = X \times Y$.

При каждом $c \in C$ имеем вероятностное пространство: $\langle D, B, P_c \rangle$, где B — σ -алгебра, P_c — вероятностная мера.

Параметр c будем называть *стратегией природы*.

Риск Решающей функцией (алгоритмом классификации) называется соответствие $\lambda: X \rightarrow Y$.

Качество принятого решения оценивается заданной функцией потерь $\mathcal{L}: Y^2 \rightarrow [0, \infty)$.

Например, $\mathcal{L}(y, y') = \begin{cases} 0, & y=y' \\ 1, & y \neq y' \end{cases}$.

Под риском будем понимать средние потери:

$$R(c, \lambda) = E\mathcal{L}(y, \lambda(x)) = \int_D \mathcal{L}(y, \lambda(x)) P_c(dx, dy),$$

$x \in X, y \in Y$.

Цель — построить решающую функцию, которая бы минимизировала риск, но риск зависит от распределения, которое неизвестно.

Метод построения решающих функций Пусть $Q: D^N \rightarrow \Lambda$ — метод (алгоритм) построения решающих функций, $\lambda_{Q,S}$ — функция, построенная по выборке S методом Q , Λ — заданный класс решающих функций.

Требуется найти метод, который бы минимизировал средний риск:

$$F(c, Q) = \mathbb{E}_{D^N} R(c, \lambda_{Q,S}).$$

Неочевидным образом, эта задача оказывается близка проверке статистических гипотез.

7.3.3 Связь с задачей проверки гипотез

Самые первые методы машинного обучения как правило проверяли гипотезу (например нормальности), и в соответствии с ней строили решение.

Но уже Фишер предложил метод, не делающий предположений о распределении.

- Классическая статистическая гипотеза: распределение принадлежит заданному классу.
- Статистическая гипотеза в машинном обучении: распределение таково, что заданный метод построения решающих функций обеспечивает требуемое качество.

Статистические методы принято разделять на параметрические и непараметрические. Методы машинного обучения большей частью находятся вне этой классификации, поскольку большинство методов не пытаются оценить полную вероятностную модель.

7.3.4 Байесовский подход

Часто статистические подходы к машинному обучению называют байесовской теорией. Это крайне неудачная терминология.

Настоящий байесовский подход подразумевает задание априорных вероятностей на гипотезах.

«Парадокс конвертов»

Игроку предлагается выбрать один из двух одинаковых на вид запечатанных конвертов с деньгами, причём известно, что сумма в одном из них в 10 раз больше, чем в другом. При этом игроку разрешается вскрыть один конверт, после чего решить, забрать его или оставшийся запечатанным.

Пусть в первом конверте оказалось x рублей.

Если считать, что во втором конверте может быть равновероятно $10x$ или $\frac{x}{10}$, то математическое ожидание выигрыша при выборе второго конверта будет $5.05x$. Но это противоречит здравому смыслу.

Парадокс является классическим примером некорректного использования байесовского подхода.

Вероятностная модель должна допускать симуляцию.

Вероятностная интерпретация задач машинного обучения во все не подразумевает использования байесовского подхода. И она никак не ограничивает исследователя в выборе решаемых задач, т.е. может использоваться в любых задачах машинного обучения (хотя есть задачи, где это нецелесообразно).

7.3.5 Случайность и Колмогоровская сложность

Обсудим вопрос, всегда ли можно использовать вероятностную интерпретацию машинного обучения.

Часто в учебниках говорится, что теория вероятностей изучает эксперименты, которые можно провести многократно в одинаковых условиях.

Однако в первом классе школы аналогичным образом говорится, что вычитать можно только из большего меньшее, а в старших классах говорится, что корень можно извлекать лишь из неотрицательных чисел.

Дальнейшее знакомство с математикой подталкивает к мысли, что в этой дисциплине всё, что нельзя, это на самом деле можно.

Точно так же и теория вероятностей далеко ушла от ограничений, при которых изначально создавалась. Так случайные процессы вполне допускают эксперименты, условия которых ни разу не повторяются.

Вопрос сводится к тому, можно ли считать имеющиеся данные случайной выборкой (не требуется независимость и однородность).

Перейдём к обсуждению того, что считать случайным.

Пример. Имеются следующие двоичные последовательности:

- 1) 111111111111111111,
- 2) 010101010101010101,
- 3) 1000010001110100011010111000111,
- 4) 0110111100101000101000011011110.

Какие из них являются случайными?

Кажется, что последовательности 1 и 2 неслучайны, а 3 и 4 — случайны.

Но чтобы ответить на вопрос, нужно дать определение случайной последовательности.

Определение. Случайная последовательность — это последовательность, которая получена в результате стохастического эксперимента (является реализацией случайного процесса или случайной функцией).

При таком определении не важно, как выглядит последовательность, важно лишь как она получена. Однако если кто-то скажет нам, что получил последовательность 1 или 2, подбрасывая монету (1 — герб, 0 — решка), то мы ему не поверим. И будем правы, скорее всего.

Но можно представить ситуацию, когда достаточно богатый и очень экстравагантный человек нанял 10000 испытателей, которым поручил под запись на камеру проводить серии по 20 бросаний монеты. Достаточно быстро он станет обладателем честного (без склеек) ролика, на котором монета выпадает гербом 20 раз. Получается, что в этом случае последовательность 1 придётся признать случайной, поскольку она действительно получена в эксперименте.

По поводу случайности последовательности 3 сомнений не воз-

никает... пока не будет раскрыта информация, что она является двоичным представлением числа, которое в десятичной записи выглядит как 1111111111.

Аналогично, псевдослучайные последовательности от генераторов случайных чисел случайными никак не являются, но по их виду этого не понять.

Если бы автор заявил, что последовательность 4 он получил, бросая монету, то опровергнуть это утверждение было бы невозможно. Известно, что последовательности, которые человек пишет «из головы», имитируя случайность, можно достаточно надёжно отличить от реальных повторных независимых испытаний, но для этого последовательность нужна намного длиннее.

Однако автор признался, что последовательность 4 он составил, имитируя случайность. Тем не менее, её действительно можно считать случайной. Это не были независимые испытания. Но это реализация случайного процесса (пожалуй, точнее будет термин случайная функция) неизвестного вида с неизвестными параметрами. И этот процесс можно исследовать (и такие исследования проводились).

Теперь если кто-то предъявит последовательность 4 и скажет, что получил её бросанием монеты, то мы ему поверим? Конечно нет. Мы ведь будем (практически) точно знать, что он переписал эту последовательность из данной статьи.

Итак, термин случайная последовательность в литературе используется в двух принципиально разных смыслах: как последовательность, полученная как реализация случайного процесса (стохастического эксперимента) и как последовательность, выглядящая случайной, т.е. не имеющая видимых закономерностей.

Это не разные интерпретации случайности, это содержательно разные понятия, которые не исключают друг друга, а дополняют.

Поскольку в одной работе нельзя один термин использовать в разных смыслах, мы будем под случайностью понимать первое понятие и называть случайным то, что имеет вероятность.

При этом второе понятие очень важно и формализуется через алгоритмическую (Колмогоровскую) сложность.

В ряде карточных игр есть правила, требующие делать пере-

сдачу, если раздача оказалась «недостаточно случайной». Это как раз пример в духе алгоритмической сложности.

Алгоритмическую сложность можно использовать как статистику для формирования критерия согласия для проверки «гипотезы случайности».

Итак, случайность определяется способом получения последовательности, но не её видом. Хотя по виду последовательности можно проверять гипотезы относительно процесса её получения.

В машинном обучении существует как вероятностная, так и невероятностные интерпретации. Выбор на усмотрение исследователя. Вероятностная картина мира является универсальной и непротиворечивой, но не единственной.

7.3.6 Классификация методов машинного обучения

Поскольку метод (алгоритм) построения решающей функции есть функция (отображение множества выборок во множество решающих функций), то методы, как функции, можно классифицировать на явные и неявные. Такая классификация имеет смысл, однако более распространённым и важным является разделение методов на методы, оценивающие распределения, и методы непосредственного построения решающих функций.

Методы, основанные на восстановлении распределений

Методы основаны на том, чтобы сначала оценить распределения, а потом построить решающую функцию, подставив оценку вместо распределения.

Традиционно в прикладной статистике принято разделять параметрические и непараметрические методы.

Параметрические методы

В параметрических методах предполагается, что оцениваемое распределение принадлежит заданному параметрическому семейству, например, классу нормальных распределений. В этом случае для восстановления распределения достаточно оценить параметры. Частным случаем параметрического подхода является байесовский подход. В этом случае на параметрах постулируется некоторое априорное распределение и по формуле Байеса вычисляется

апостериорное (при условии выборки) распределение, по которому можно строить решение.

Непараметрические методы

Данный класс методов основан на непараметрическом оценивании распределения. Пример – парзеновские оценки плотности, которые можно рассматривать как обобщение одного из базовых методов оценивания плотности — с помощью гистограммы.

Методы, конструирующие решающие правила

Методы с явным заданием решения

Примером метода из данного класса может служить дискриминант Фишера, который аналитически выражается как явная функция выборки.

К этой категории также относятся метрические методы построения решающей функции, в частности, рассмотренный ранее метод прецедентов.

Методы на основе оптимизации эмпирического критерия

Большой класс методов предполагает оптимизацию некоторого функционала качества (как правило, эмпирического риска) в некотором классе решающих функций, например метод опорных векторов, решающие деревья, нейронные сети.

7.4 Сравнение методов построения решающих функций

Метод построения решающих функций Пусть $Q: D^N \rightarrow \Lambda$ – метод (алгоритм) построения решающих функций, $\lambda_{Q,S}$ – функция, построенная по выборке S методом Q , Λ – заданный класс решающих функций.

Метод \tilde{Q} , минимизирующий эмпирический риск, есть

$$\lambda_{\tilde{Q},S} = \arg \min_{\lambda \in \Lambda} \tilde{R}(S, \lambda).$$

Сравнение методов построения решающих функций

- Выбор эталонного набора тестовых задач.

- Введение понятия оптимальности метода.

Сопоставление с задачей проверки гипотез Статистический критерий можно считать частным случаем метода построения решающих функций, когда в роли решающей функции выступает предикат.

В роли риска ошибки второго рода, но функция потерь зависит от распределения.

Известно множество статистических критериев, но нет понятия наилучшего критерия (кроме случая известной простой альтернативы).

Система «Полигон» — 1980-е [5]

Принципы:

- для каждого метода включается «эталонная» задача,
- на «своей» задаче метод должен работать лучше других,
- возможно оценить степень универсальности метода,
- тестовая единица - таблица данных.

Система «Полигон» — 2000-е [4]

Принципы:

- использование реальных задач,
- большое число характеристик качества,
- основной критерий - скользящий экзамен.

Тестовые единицы

Возможные тестовые единицы:

- таблица данных,
- распределение,
- класс распределений.

Об оптимальности методов построения решающих функций
Проблема определения оптимальности метода
Напомним, что метод — это отображение выборок в решения.

- Для таблицы данных понятие оптимального метода не имеет смысла.
- Для заданного распределения оптимальный метод вырожден — он любой выборке сопоставляет байесовское решающее правило.
- Об оптимальности метода можно говорить только для класса распределений.
- Даже для нормальных распределений оптимальный метод неизвестен.

Минимаксный подход к оцениванию качества

- Максимальное по классу распределений значение риска для всех методов одинаково.
- Вводить ограничения сверху на Байесовский уровень ошибки не имеет смысла.
- Использование максимума не абсолютного риска, а отнесённого к достижимому уровню, позволяет ввести осмысленное понятие метода, оптимального на классе распределений.

Похожая проблема в статистическом оценивании решается требованием несмещённости эффективной оценки.

Достижимый уровень качества

- Интересует не Байесовский уровень ошибки, а тот, который реально достижим.
- Необходимо задавать для каждого распределения из класса.
- Определяется на основе эталонного метода или из других соображений.

Выбор классов распределений

- Класс распределений не должен быть ни узким ни широким — иначе получаем соответственно вырожденный метод или аналог NFL.
- Представительность для исследований: все значения достижимого риска, максимально смещённые оценки.
- Параметр сложности. Универсальность.
- Замкнутость относительно допустимых преобразований пространства переменных.

Варианты классов распределений

- Класс кусочно-постоянных распределений.
- Класс нормальных распределений.
- Класс, сформированный случайными решающими деревьями.
- Ядерные функции для условных вероятностей.

7.4.1 Выводы

Выводы

- Задача машинного обучения заключается в конструировании методов построения решающих функций.
- Критерий качества первичен. Он определяет оптимальное решение.
- Необходимо различать целевой и эмпирический критерии.
- До сих пор не введено понятие, что значит: один метод классификации лучше другого.
- Задачи машинного обучения похожи на задачи проверки статистических гипотез.

7.5 Некоторые задачи машинного обучения

Перечислим типичные задачи построения статистических решений.

Распознавание образов (классификация «с учителем») — задача, которую будем рассматривать наиболее подробно.

Регрессионный анализ (восстановление зависимостей) — точнее было бы построение зависимостей.

Кластерный анализ (таксономия, автоматическая группировка, классификация «без учителя») — рассматривать не будем. Пожалуй на оценивание смеси распределений.

Задача упорядочивания объектов — частный случай классификации, когда оцениваем вероятность принадлежности классу и упорядочиваем по вероятности.

Задача обнаружения закономерностей — нет целевой переменной. Похоже на кластеризацию, но более общая.

Прогнозирование временных рядов — можно сводить к другим задачам, но есть специализированные подходы.

Планирование эксперимента — обобщение задачи регрессии на случай, когда можем выбирать точки.

Поиск глобального экстремума — частный случай планирования эксперимента, когда нужно аппроксимировать не всю функцию, а только найти максимум.

Анализ клиентских сред — выявление связей между объектами, например, покупателями и товарами.

Тематическое моделирование — классификация текстов по темам.

Генеративные модели — сейчас наиболее популярны. По сути специфическая задача. Но большие текстовые модели позволяют решать и другие задачи.

Это задачи машинного обучения. Есть другой подход (сейчас в тени) — алгоритмический, экспертные системы, символьные вычисления.

Глава 8

Некоторые задачи анализа данных

Методы глобальной оптимизации, СПА, теорема о небесплатном завтраке, кластерный анализ, временные ряды

8.1 Методы глобальной оптимизации

Методы оптимизации — отдельная область математики, которая имеет определённую связь с анализом данных (неспроста на kaggle проводились конкурсы по решению оптимизационных задач).

В первую очередь, эта связь заключается в том, что большинство методов анализа данных основаны на решении оптимизационных задач и используют методы оптимизации.

Однако в некоторых постановках сама задача оптимизации оказывается по духу ближе к области машинного обучения, чем к области математического программирования. Дело в том, что задачи математического программирования предполагают выполнение некоторых строгих ограничений на целевую функцию (например, гладкость или выпуклость). В отсутствие строгих предположений классические методы оптимизации неприменимы.

8.1.1 Задача поиска глобального экстремума

Под экстремумом понимается минимальное или максимальное значение функции. Без ограничения общности можно считать, что всегда ищется максимум, поскольку, если требуется найти минимум, то можно вместо исходной функции рассматривать функцию, умноженную на -1 , у которой максимум достигается в точке минимума исходной функции.

О поиске глобального экстремума имеет смысл говорить, когда функция является многоэкстремальной, т.е. имеет более одного локального экстремума. Локальным максимумом называется точка, в которой значение функции больше, чем в любой точке некоторой её окрестности.

Одним из наиболее наглядных примеров многоэкстремальной функции является рельеф местности, т.е. зависимость высоты над уровнем моря от географических координат точки.

Задача поиска глобального экстремума (или задача оптимизации) заключается в следующем. Требуется выбрать последовательность точек в пространстве X так, чтобы среди значений функции в выбранных точках оказалось значение, близкое к максимальному. При этом выбор последующих точек зависит от значений функции в точках, выбранных ранее.

Другими словами, задана некоторая функция, которая скрыта. Исследователь может выбирать любые точки из области определения и узнавать значение функции в них. Задача заключается в том, чтобы, использовав как можно меньшее число точек, обнаружить как можно большее значение функции.

Для решения задач многоэкстремальной оптимизации наиболее часто используется подход, основанный на идее адаптивного поиска. Данная идея заключается в том, что точки выбираются в соответствии с некоторой мерой «перспективности» областей пространства X . При этом «перспективность» области тем выше, чем большие значения функции в ней уже обнаружены и чем «менее исследована» область, т.е. чем меньше точек в ней поставлено ранее.

Идеи адаптивного поиска также развиваются в эволюционных (в т.ч. генетических) алгоритмах, а также в алгоритмах типа му-

равыиной колонии.

Задача поиска глобального экстремума может рассматриваться как частный случай задачи планирования эксперимента. Задача планирования эксперимента это особая модификация задачи восстановления зависимостей, когда таблица данных не задана изначально, а формируется при участии исследователя, который может выбирать точки пространства X , для которых будет сообщены значения целевой функции. Требуется выбирать точки таким, образом, чтобы как можно точнее оценить целевую функцию. Критерием качества такой оценки обычно выступает средний квадрат отклонения. Требование найти экстремум функции может считаться одним из вариантов критерия качества восстановления зависимости.

8.1.2 Формальная постановка

Сформулируем задачу глобальной оптимизации.

Пусть Ψ – некоторый класс функций с общими областью определения Z и областью значений R , т.е. $\psi : Z \rightarrow R$, $\psi \in \Psi$. Множество R упорядочено.

Алгоритмом (детерминированным алгоритмом) поиска экстремума будем называть множество (последовательность) функций

$$A = (A_i, i = 1, \dots, L), \quad A_i : (Z \times R)^{i-1} \rightarrow Z,$$

где параметр L определяет число точек, которые можно использовать при поиске экстремума. При этом A_1 – константа (из Z).

Результатом применения алгоритма A к функции ψ будет последовательность

$$A | \psi = ((z_i, r_i), i = 1, \dots, L),$$

где

$$z_i = A(z_1, r_1, \dots, z_{i-1}, r_{i-1}), \quad r_i = \psi(z_i).$$

Таким образом, первая точка фиксирована, а остальные ставятся в зависимости от полученных значений оптимизируемой функции.

Стохастическим (случайным) алгоритмом поиска экстремума будем называть совокупность семейств вероятностных мер

$$S = (\mathsf{P}_i(dx \mid \alpha_i), i = 1, \dots, L), \quad \alpha_i \in (Z \times R)^{i-1}.$$

Все меры заданы на некоторой σ -алгебре подмножеств пространства Z .

Таким образом, первая точка выбирается в соответствии с фиксированным вероятностным распределением, для остальных вероятностная мера корректируется в зависимости от полученных значений оптимизируемой функции.

Чтобы задача поиска глобального экстремума была поставленной, нужно задать критерий качества алгоритма оптимизации, как функцию $K = \Psi \times \Lambda \rightarrow [1, \infty)$, где Λ – множество всех детерминированных и стохастических алгоритмов. В настоящее время такая задача решена только для некоторых частных постановок. Задача сравнения эффективности алгоритмов оптимизации и выбора алгоритма, наиболее подходящего для заданного класса задач, остается открытой.

8.1.3 Теорема о небесплатном завтраке

Теорема о небесплатном завтраке (или NFL, т.е. No Free Lunch theorem) является очень известным результатом. Приведём один из простейших её вариантов.

Пусть область определения Z и область значений R целевой функции являются конечными множествами, каждое из N элементов. Также положим, что целевая функция во всех точках принимает различные значения. Тогда общее число всех функций равно $N!$.

Критерий качества для результата оптимизации определим очень просто: будем считать оптимизацию успешной, если не более чем за M шагов найден глобальный экстремум.

Теорема. Для любого алгоритма оптимизации количество функций, которые он успешно оптимизирует, одинаково.

Иными словами, все алгоритмы равнозначны, и самый «интеллектуальный» алгоритм оптимизации не лучше самого «глупого» (например, расстановки точек наугад).

Обычно данная теорема интерпретируется как утверждение, что если не делать вообще никаких предположений об оптимизируемой функции, то и разумного алгоритма оптимизации предложить нельзя.

Попробуем построить подобный пример для задачи машинного обучения.

Предположим, что у нас есть некоторое устройство, которое генерирует последовательность длины N , состоящую из 0 и 1. Всего возможно 2^N различных последовательностей. Требуется по первым M значениям предсказать следующее значение. Алгоритмом предсказания назовём функцию, которая каждой последовательности из M нулей и единиц сопоставляет 0 или 1.

Легко доказать, что для всех таких алгоритмов предсказания число последовательностей, которые они успешно предсказывают, будет одинаково (и равно 2^{N-1}).

Значит ли это, что все алгоритмы предсказания одинаковы по качеству?

Кажется, что так не должно быть. Ведь если к некоторому моменту мы наблюдаем, например, 90 единиц и 10 нулей, то разумным будет предсказать в качестве следующего значения 1, а не 0.

Получается, что в машинном обучении есть полный аналог NFL, но из него не следует равноточности любых методов прогнозирования.

Всё дело в том, что NFL не учитывает мощнейшего инструмента, который использует машинное обучение (иногда явно, но чаще неявно), а именно механизм проверки гипотез (или доверительных интервалов).

Кроме того, благодаря парадоксу конвертов мы выяснили, что если мы не имеем информации о вероятностях гипотез, то это не даёт нам право считать эти гипотезы равновероятными. А NFL именно так и делает (даже если явно вероятности и не вводит).

Сравнивать количество целевых функций или количество последовательностей не имеет никакого смысла, хотя бы потому что их множество может быть континуально. Имеет смысл сравнивать вероятности успешного решения задачи для разных методов (ал-

горитмов).

В духе NFL мы можем задать равномерное распределение на всех последовательностях. Но при таком распределении вероятность появления 1 на любом месте последовательности равна 0,5. Однако если, как в примере, из 100 значений мы получили 90 единиц, то это даёт основания отвергнуть гипотезу о том, что вероятность единицы равна 0,5. В результате мы вполне можем заключить, что с некоторой доверительной вероятностью точность нашего алгоритма прогнозирования превышает 0,5, и построить доверительный интервал для этой точности (подобное уже делали).

Таким образом «парадокс» NFL снимается.

Современные методы машинного обучения не делают предположений о виде распределений, но неявно проверяют гипотезу о том, что решение заданной формы имеет приемлемое качество.

Таким образом, NFL действительно следует интерпретировать как тот факт, что без всяких предположений решать задачи анализа данных невозможно. Но любые разумные методы такого рода предположения и делают, и проверяют — чаще неявно.

8.1.4 Алгоритм СПА

Алгоритм случайного поиска с адаптацией [7] является представителем стохастических алгоритмов оптимизации. Для использования алгоритма СПА необходимо чтобы на точках из Z была определена мера сходства, которую будем для удобства называть расстоянием, при этом не подразумевая обязательного выполнения свойств метрики. Идея алгоритма СПА состоит в том, что плотности вероятности для выбора последующих точек задаются так, чтобы их значения были больше в окрестностях тех из уже поставленных точек, в которых значения целевой функции ближе к оптимальному, а также в тех областях, где плотность точек (экспериментов) ниже (в «малоизученных» областях). При этом данные плотности зависят от числа уже проведенных экспериментов.

Эта идея может реализовываться (конкретизироваться) различными способами. Первая версия алгоритма была разработана

для пространства бинарных переменных. Следующие версии использовали для разбиения пространства поиска деревья решений.

8.2 Кластерный анализ

Задачи кластерного анализа состоят в том, чтобы разбить заданную совокупность объектов на группы так, чтобы внутри каждой группы объекты были в некотором смысле похожими, а объекты разных групп максимально отличались. Синонимами кластерного анализа являются таксономия, автоматическая группировка, классификация «без учителя». Последний термин подразумевает, что в отличие от рассматривавшейся ранее задачи классификации («с учителем») в обучающей выборке отсутствует информация о принадлежности объектов классам (т.е. о значениях переменной Y).

Таким образом, обучающая выборка в задаче кластерного анализа имеет вид

$$S = (x^i, i = 1, \dots, N).$$

Иногда в значении кластерного анализа используют термин «автоматическая классификация», однако данный термин лучше не использовать ввиду неоднозначности, поскольку классификация «с учителем» имеет такие же основания называться автоматической, поскольку производится компьютерной программой.

Кластеризация обычно использует некоторую функцию сходства объектов, например, расстояние. Некоторые методы ограничиваются понятием соседства объектов в топологическом смысле, дополнительно используя некоторую меру

Первый подкласс методов — методы «нисходящей» кластеризации. В этих методах выбирается количество кластеров и производится разбиение множества объектов на кластеры (группы объектов) в соответствии с некоторым критерием.

8.2.1 Метод k -средних

Метод заключается в разбиении объектов на группы так, чтобы сумма расстояний от объектов до центров групп была мини-

мальной.

8.2.2 Оценивание смеси распределений

Смесью распределений называется распределение, функция распределения которого представляется взвешенной суммой некоторых функций распределения заданного вида. Чаще смесь задаётся через плотности вероятности

$$\varphi(x) = \sum_{j=1}^k \alpha_j \varphi_j(x),$$

где $\varphi_j(x)$ – плотности заданного вида, α_j – весовые коэффициенты.

Плотности обычно выбираются из некоторого параметрического семейства. В этом случае задача кластеризации становится стандартной статистической задачей оценивания параметров распределения. Для её решения существуют хорошо развитые методы. В кластерном анализе чаще всего используются смеси нормальных распределений.

8.2.3 Иерархическая кластеризация

Данный класс методов использует идею «восходящей» кластеризации, когда точки последовательно присоединяются к кластерам, а кластеры последовательно укрупняются. При этом последовательное объединение кластеров формирует дерево.

На рис. 8.1 приведён пример иерархической кластеризации. Справа приведено дерево кластеризации. Два поддерева, выходящие из корневой вершины, в точности совпадают с кластерами, полученными методом k -средних при $k = 2$. При этом дерево несёт гораздо больше информации о структуре расположения точек, чем обычная кластеризация.

Иерархическая кластеризация может использоваться и в том случае, когда известны только попарные расстояния между точками, а координаты точек неизвестны или вовсе не имеют смысла. Такая ситуация может возникнуть, например, при группировке музыкальных произведений. С помощью экспертов могут

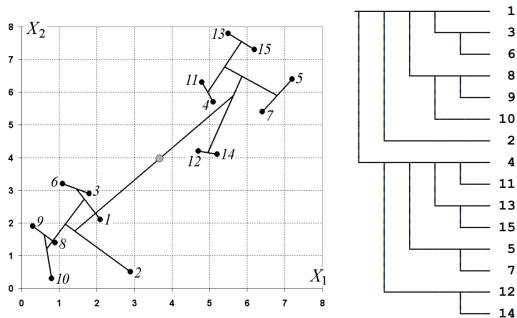


Рис. 8.1: Пример иерархической кластеризации.

быть даны численные оценки похожести (или, наоборот, отличия) произведений. Тогда для каждого произведения будут известны «расстояния» от него до каждого из остальных произведений. При этом музыкальные произведения не задаются точками в пространстве переменных.

8.3 Анализ временных рядов

Прогнозировать временной ряд можно путём сведения к таблице данных.

Также есть специализированные методы (Хольта–Винтерса, ARIMA и т.п.), есть специальные архитектуры нейронных сетей (рекуррентные, например, LSTM).

Задачу прогнозирования временного ряда проиллюстрируем для случая одной бинарной (двоичной) переменной. Для её решения будем использовать одну из модификаций метода прецедентов, сводящуюся в данном случае к оцениванию частот различных предысторий.

В качестве исходных данных возьмём последовательность, сформированную вручную из 0 и 1, выбиравшихся наугад. На первый взгляд, может показаться, что вероятности появления 0 и 1 не зависят от ранее выбранных значений и равны 0,5.

Если бы это было так, то такую последовательность не имело бы смысла прогнозировать, поскольку для любого решающего

правила вероятность ошибки была бы 0,5. Однако человек, вообще говоря, не является идеальным генератором случайных чисел и, даже стараясь выбирать числа равновероятно и «бессистемно», он составляет ряд, в котором присутствуют статистические закономерности.

Исходная длина последовательности, представленной в таблице, равна 150. Для того, чтобы иметь возможность проверить качество прогноза, разобьём ряд на две части: обучающую, длиной 100, и контрольную, длиной 50.

Таблица. Временные ряды

обучение

01101000101001101110101100001011100111101010110101

000010110111110101011110101010010111010101101010

контроль

1110101011100110000010110101001011100101010101011

прогноз

101110101011011111011010111011111010101010

Выберем значения параметра d – длины предыстории, по которой будет производиться прогноз. При используемом методе прогнозирования d должно быть таким, чтобы 2^d – число возможных предысторий было существенно меньше длины обучающей последовательности. Возьмём $d = 4$.

Далее перебираем все возможные последовательности из пяти 0 и 1 и подсчитываем, сколько раз они встречаются в обучающем ряде. Например, последовательность 0000 не встретилась ни разу, а фрагмент 00001 присутствует два раза.

Сравнив прогноз с контрольной последовательностью, обнаруживаем 17 несовпадений, т.е. доля ошибок прогноза есть 0,34.

Получившаяся доля ошибок существенно отличается от 0,5.

Применив классические критерии проверки гипотез, легко показать, что это отличие статистически значимо и подобный ряд действительно можно прогнозировать с вероятностью ошибки, меньшей 0,5.

8.4 Прогнозирование нескольких переменных

Рассмотрим следующую задачу прогнозирования.

Пусть имеется генеральная совокупность объектов Γ , для которой определена произвольная вероятностная мера P . Каждый объект может быть охарактеризован значениями $x_1, \dots, x_n; y_1, \dots, y_m$ переменных $X_1, \dots, X_n; Y_1, \dots, Y_m$. Данные переменные могут быть произвольных типов (вещественные, целые, порядковые, номинальные, бинарные).

Рассматриваемая задача состоит в том, чтобы для произвольного объекта из Γ по известным значениям переменных X_1, \dots, X_n предсказать значения переменных Y_1, \dots, Y_m на основе анализа имеющейся эмпирической информации. Заметим, что задачи построения решающей функции распознавания и регрессионной функции являются частным случаем рассматриваемой задачи.

Обычно в подобных случаях все переменные прогнозируют отдельно, т.е. независимо.

Но целевые переменные могут быть объективно зависимыми (даже условно, т.е. при условии конкретных x). Поэтому лучше прогнозировать переменные последовательно, добавляя уже предсказанные переменные в модель.

8.5 Использование экспертных знаний

Высказывание как случайное событие.

В работе [8] предлагается метод решения задачи прогнозирования на основе анализа эмпирической информации, которая может быть представлена либо набором высказываний экспертов, либо таблицей экспериментальных данных (выборкой), либо экспертными высказываниями и таблицей данных одновременно.

Чтобы распространить байесовский подход на случай, когда эмпирическая информация представлена экспертными суждениями, предлагается статистическая интерпретация экспертной информации. При этом предполагается, что экспертная информация представлена высказываниями нескольких экспертов. Допускается, что высказывания различных экспертов могут быть несогла-

сованы и, в-частности, быть противоречащими друг другу в той или иной степени, совпадающими, частично совпадающими, дополнительными.

Одним из достоинств предлагаемой в работе реализации байесовского подхода к решению поставленной задачи является использование слабых гипотез относительно распределений.

8.5.1 Постановка задачи

Рассмотрим задачу прогнозирования целевой переменной Y со значениями $\{-1, 1\}$.

Введем обозначение

$$g(x) = \frac{\mathbb{P}(dx, y = 1)}{\mathbb{P}(dx)}.$$

Таким образом, $g(x)$ представляет собой вероятность первого класса при условии, что значения переменных X_1, \dots, X_n равны $x = (x_1, \dots, x_n)$.

Под задачей прогнозирования будем понимать восстановление $g(x)$ на основе эмпирической информации, то есть построение некоторой оценки $\bar{g}(z)$ функции $g(z)$.

Для удобства дальнейших обозначений введем множество C , элементы которого будем называть стратегиями природы и обозначать c . При этом каждой функции $g(z)$ поставим во взаимно-однозначное соответствие некоторую $c \in C$. В дальнейшем вместо $g(x)$ будем записывать $g_c(x)$.

В отличие от известных подходов к решению задач восстановления распределений на основе выборки предлагаемый в данной работе метод ориентирован на использование эмпирической информации как в виде таблицы экспериментальных данных, так и в виде набора высказываний, полученных от многих экспертов.

Формально высказывание эксперта может быть представлено в виде следующей тройки:

$$B_i = \langle E^i, \beta^i, \gamma_i \rangle,$$

где $i = \overline{1, N}$ (N – число высказываний, полученных от экспертов). При этом $E^i \subseteq D_X$, $\beta_k^i \in [0, 1]$, $\gamma_i \in [0, 1]$.

Данное высказывание может быть проинтерпретировано следующим образом: «Если x – некоторая (соответствующая объекту) точка из D_X , то вероятность того, что $y == 1$ будет равна β^i ». Число γ_i отражает уверенность эксперта в истинности сделанного утверждения. Заметим, однако, что смысл высказывания имеет для нас значение лишь с точки зрения его влияния на зависимость вероятности появления данного высказывания от стратегии c .

Предположим, что для каждой стратегии c нам известна $P(B_i/c)$ – условная вероятность появления высказывания B_i . Тогда, поскольку эксперты делают высказывания независимо, то $P(B/c) = \prod_{i=1}^N P(B_i/c)$. Здесь $B = \{B_i | i = \overline{1, N}\}$ – множество всех высказываний, представленных экспертами.

Если помимо экспертной информации мы имеем и обычную таблицу экспериментальных данных, то мы можем просто перемножить вероятность появления экспертной информации и вероятность появления выборки.

8.5.2 Байесовский подход

Чтобы воспользоваться байесовским подходом к восстановлению $g(x)$, нужно задать априорную вероятностную меру $\mathsf{P}(C)$. После этого

$$\bar{g}^*(x) = \frac{\int_C g_c(x) P(B/c) \mathsf{P}(dc)}{\int_C P(B/c) \mathsf{P}(dc)}.$$

Данное выражение, полученное из формулы Байеса, дает оптимальную оценку для функции $g(x)$ в том смысле, что $\bar{g}^*(x)$ есть условное (при заданном B) математическое ожидание функции $g(x)$.

Алгоритм, который на основе имеющейся информации B строит оценку функции $g(x)$ в соответствии с формулой, будем называть байесовским алгоритмом решения рассматриваемой задачи прогнозирования.

Обычно в теории статистических решений при задании априорной меры $\mathsf{P}(C)$ вводятся сильные ограничения на класс страте-

тегий природы, которые во многих случаях не оправданы. Однако существуют универсальные классы распределений, на которых можно вводить разумную априорную меру.

8.5.3 Статистическая интерпретация экспертной информации

В отличие от известных методов логического анализа экспертных высказываний в данной работе предлагается статистическая интерпретация экспертной информации. При этом полагается, что вероятность появления каждого высказывания зависит от того, какая стратегия природы имеет место.

Эксперта s будем представлять как оператор, сопоставляющий каждой стратегии c условное распределение вероятностей $P_s(u/c)$ на множестве U , где U – множество подмножеств всех допустимых высказываний. Заметим, что U – конечно ввиду конечности алфавита, в котором записываются высказывания, и естественных ограничений на длину и количество высказываний. Если эксперты, делающие высказывания, случайно выбираются из множества S в соответствии с некоторой вероятностной мерой $P(S)$, то $P(u/c) = \sum_s P_s(u/c)P(s)$.

Заметим, что функция $P(u/c)$ является «в природе» вполне определенной, а потому ее конкретный вид может быть установлен непосредственным экспериментальным исследованием.

8.5.4 Примеры

Получаемую в результате оценку $\bar{g}^*(x)$ будем записывать также в форме набора высказываний, которые будут иметь вид: $\langle \tilde{E}_l, \tilde{\beta}_l \rangle$, где $\tilde{E}_l \subseteq D_X$, $\bar{g}^*(x) = \tilde{\beta}_l$, при $x \in \tilde{E}_l$; $l = \overline{1, \tilde{N}}$, где \tilde{N} – число областей постоянства функции $\bar{g}^*(x)$ в пространстве D_X .

Пример 1. Пусть исходная информация представлена единственным высказыванием:

$$1) \langle E, 0.9, 0.8 \rangle.$$

Применяя соотношение (1), получаем:

$$1) \langle E, 0.86 \rangle.$$

Видим, что $\tilde{\beta}$ оказалась не равной β , а сдвинутой к 0.5 (т. е. в сторону меньшей определенности). Это вполне оправдано, поскольку уверенность эксперта (как и доверие ему) не абсолютна.

Пример 2. Пусть даны два одинаковых высказывания двух экспертов:

- 1) $\langle E, 0.9, 0.8 \rangle$;
- 2) $\langle E, 0.9, 0.8 \rangle$.

В результате получим:

- 1) $\langle E, 0.89 \rangle$.

Оценка $\tilde{\beta}$ уже гораздо ближе к β_1 (или к β_2), чем в первом случае. Это означает, что степень доверия двум экспертом, независимо подтверждающим мнения друг друга, выше, чем доверие высказыванию одного эксперта.

Пример 3. Даны информация:

- 1) $\langle \{a, b\}, 0.9, 0.8 \rangle$;
- 2) $\langle \{b, c\}, 0.1, 0.8 \rangle$;
- 3) $\langle \{c, d\}, 0.9, 0.8 \rangle$.

В данном примере пространство D_X состоит из значений $\{a, b, c, d\}$ единственной номинальной переменной X .

Результат согласования:

- | | |
|--------------------------------------|--------------------------------------|
| 1) $\langle (x = a), 0.8 \rangle$; | 2) $\langle (x = b), 0.72 \rangle$; |
| 3) $\langle (x = c), 0.72 \rangle$; | 4) $\langle (x = d), 0.8 \rangle$. |

Результат представляется интуитивно вполне оправданным. Действительно, поскольку высказывания противоречат друг другу, то оценки для $g(x, 0)$ значительно сдвинуты к 0.5. Далее, ввиду того, что второй эксперт противоречит одновременно как первому так и третьему, то доверие его утверждению относительно ниже. Этим, в частности, объясняется, например, то, что для точки b решение ближе к оценкам первого эксперта, чем второго.

Глава 9

Проблемы искусственного интеллекта

На ветке дерева, под солнцем ярким,
Тукан разноцветный весело кричит.
В красных перьях блеск, в глазах огоньки,
Он рядом с бананом, своим друзьями.

Рядом с ними утконос на воде плавает,
С кловом длинным и странным на водоросли
налегает.
Он клюет рыбок и ракушек много,
Утконос весело шныряет в пруду глубоком.

Вдруг пылесос вылез из травы,
Сказал он: «Пора подметать мне снег и пыль».
Он быстро крутится, весь в пыли и грязи,
Но никто не замечает его усилий.

Тукан, банан, утконос и пылесос,
Все они живут в этом мире разномасле.
Но каждый из них важен и нужен,
Ведь каждый свое дело исполняет с усердием.

ChatGPT-3.5 turbo

Изначально автор планировал построить логику изложения материала главы в соответствии с исторической последовательностью. Но тогда у читателя мог возникнуть резонный вопрос: «Зачем изучать какие-то устаревшие подходы, если все задачи

решают ChatGPT и мультимодальные модели?» Пожалуй, у нас нет иного выбора, кроме как начать обсуждение вопросов искусственного интеллекта с ChatGPT.

9.1 Большие языковые модели

9.1.1 Тестирование возможностей

Будем тестировать ChatGPT-3.5 turbo. Цель — найти границы возможностей для больших языковых моделей. Для этого будем искать (придумывать) максимально простые задачи разных типов, на которых бы сеть ошибалась.

Самый простой тест — это, пожалуй, арифметические вычисления.

Вычисли выражение: $40551147 + 98005908 - 1129933$

Результат вычисления выражения $40551147 + 98005908 - 1129933$ равен 137426122.

Правильный результат: 137427122. Отличие в одной цифре, но это ошибка.

Получается, что языковые модели даже простую арифметику не могут вычислить? На самом деле, если бы бот вообще не делал арифметических ошибок, это было бы гораздо менее впечатляющее. Просто в этом случае было бы очевидно, что бот пользуется калькулятором. Но раз случаются ошибки, значит калькулятор не используется. При этом ошибки наблюдаются лишь в примерах с достаточно большими числами. Отличающееся на несколько цифр выражение $4051147 + 9800598 - 1129933$ уже было вычислено без ошибок. И то, что это сделано только лишь нейросетью, весьма удивительно, поскольку сеть обучалась на текстах.

Проверим на решении простых задач.

Задача. На поляне сидят три мухи, два утконоса, 5 мышей, три рака и 2 муравья. Сколько всего насекомых сидит на поляне?

На поляне сидит 3 мухи, 2 утконоса, 2 муравья, итого 7 насекомых.

Здесь с арифметикой порядок, но ошибка в классификации животных.

А сколько теплокровных животных было на поляне?

На поляне было 3 мухи, 2 утконоса, 5 мышей, 3 рака и 2 муравья. Все эти животные являются холоднокровными, так как теплокровными считаются только позвоночные животные. Таким образом, на поляне не было теплокровных животных.

Как видим, в ответах бывают грубые фактические ошибки.

Задача. Графиня с вероятностью 0,5 спрятала сокровища в одном из 12 стульев. Отец Фёдор разрубил 11 стульев, но сокровищ не нашёл. Какова вероятность, что сокровища находятся в последнем 12 стуле?

Для решения данной задачи, мы можем использовать условную вероятность.

Исходя из условия, вероятность того, что сокровища находятся в каждом стуле равна $1/12$, так как графиня равновероятно могла спрятать их в любом из 12 стульев.

После того, как отец Фёдор разрубил 11 стульев и не нашёл сокровища, остался только 1 стул, в котором сокровища могут находиться. Таким образом, вероятность того, что сокровища находятся в последнем 12 стуле равна $1/1$ или просто 1.

Таким образом, вероятность того, что сокровища находятся в последнем 12 стуле, при условии, что 11 стульев были разрублены и сокровища не были найдены, равна 1.

Как видим, решение полностью игнорирует утверждение условия, что вероятность 0,5 приходится на все стулья.

Эта задача относительно простая. Для вузовского курса она является типовой в теме про условные вероятности. Её можно решить и в рамках школьных знаний, если провести некоторые рассуждения. Подсказка будет дана в конце раздела.

Найди наименьшее отрицательное целое число, которое без остатка делится на 2 и на 3.

Наименьшее отрицательное целое число, которое без остатка делится и на 2, и на 3, можно найти, умножая их наименьшие общие кратные (НОК) данных чисел. $\text{НОК}(2, 3) = 6$

Таким образом, наименьшее отрицательное число, которое без остатка делится и на 2, и на 3, равно -6 .

Несмотря на то, что решение ошибочно, рассуждения выглядят неплохо. Очевидно, сеть не часто встречала вопросы «с подвохом», когда просят найти то, чего не существует.

Перейдём к логическим (или скорее алгоритмическим) задачам.

Есть задача. На столе стоят 5 коробок, пронумерованные числами от 1 до 5. В каждой коробке лежит столько спичек, каков её номер. В каждую коробку добавляем по спичке. Потом пересыпаем спички из коробки 2 в коробку 4. Затем половину спичек из коробки 4 переносим в коробку 1. Сколько спичек теперь лежит в каждой коробке?

Давайте решим задачу шаг за шагом:

Изначально в каждой коробке лежит столько спичек, каков её номер:

Коробка 1: 1 спичка

Коробка 2: 2 спички

Коробка 3: 3 спички

Коробка 4: 4 спички

Коробка 5: 5 спичек

Затем мы добавляем по одной спичке в каждую коробку:

...

Затем мы пересыпаем спички из коробки 2 в коробку 4:

...

И наконец, переносим половину спичек из коробки 4 в коробку 1:

...

Итак, после всех операций в каждой коробке лежит следующее количество спичек:

Коробка 1: 6 спичек

Коробка 2: 0 спичек

Коробка 3: 4 спички

Коробка 4: 4 спички

Коробка 5: 6 спичек

В тексте решения сокращены перечисления состояний коробок. Как видим, решение правильное.

Похоже, что задача совсем не оригинальная, и подобные встречались в обучении.

Кажется, что придумать задачу такую, чтобы похожих не было в обучении, очень трудно. Например, автор придумал слово «крокодиломорф», чтобы проверить способность сети интерпретировать слово на основе смысла морфем. Но оказалось, что такое слово широко используется (в узких кругах) и обозначает таксон, которому принадлежат крокодилы.

Задача. Имеется корзина. Операция ныкать заключается в добавлении одного ништяка в корзину. Операция тырить состоит в извлечении двух ништяков из корзины. Операция хомячить заключается в выполнении трёх операций ныкать с последующим извлечением одного ништяка из корзины. Утконос выполнил следующую последовательность операций: хомячить, тырить, ныкать, тырить, ныкать, ныкать. Каков результат этих действий, если изначально корзина пуста?

Давайте посчитаем результат последовательности операций:

1. Хомячить: выполнение трех операций ныкать и извлечение одного ништяка.

— Корзина: +3 ништяка

2. Тырить: извлечение двух ништяков из корзины.

— Корзина: +3 ништяка – 2 ништяка = +1 ништяк

3. Ныкать: добавление одного ништяка в корзину.

— Корзина: +1 ништяк + 1 ништяк = +2 ништяка

4. Тырить: извлечение двух ништяков из корзины.

— Корзина: +2 ништяка – 2 ништяка = 0 ништяков

5. Ныкать: добавление одного ништяка в корзину.

— Корзина: 0 ништяков + 1 ништяк = +1 ништяк

6. Ныкать: добавление одного ништяка в корзину.

— Корзина: +1 ништяк + 1 ништяк = +2 ништяка

Итак, после выполнения указанной последовательности операций утконосом, в корзине окажется 2 ништяка.

Как видим, решение ошибочно, поскольку модель не смогла правильно интерпретировать содержание операции «хомячить». Вероятно, для неё проблема интерпретировать составные операции.

А если при изначально пустой корзине выполнить операции: ныкать, тырить, ныкать, то каков будет результат?

Если при изначально пустой корзине выполнить операции: ныкать, тырить, ныкать, то результат будет следующим:

1. Ныкать: добавление одного ништяка в корзину.

— Корзина: +1 ништяк

2. Тырить: извлечение двух ништяков из корзины.

— Корзина: +1 ништяк – 2 ништяка = -1 ништяк

3. Ныкать: добавление одного ништяка в корзину.

— Корзина: -1 ништяк + 1 ништяк = 0 ништяков

Таким образом, после выполнения указанной последовательности операций утконосом, в корзине окажется 0 ништяков.

Ожидаемо, сеть «не догадалась», что чем бы ни были «ништяки», отрицательное их количество не может лежать в корзине.

Поставим задачу по программированию.

Напиши программу на Питоне. Программа должна вывести все двузначные числа, каждая цифра которых делится на 3.

Модель выдала корректную программу и после дополнительной просьбы поместить результат в список, используя генератор, был выдан результат.

```
numbers = [i for i in range(10, 100) if i % 3 == 0 and i / 10 %
3 == 0 and i % 10 % 3 == 0]
print(numbers)
```

Тут придаться не к чему.

А какой результат выдаст эта программа?

Эта программа выдаст список всех двузначных чисел, каждая цифра которых делится на 3. Например, результатом будет список [33, 36, 39, 63, 66, 69, 93, 96, 99].

Слово «например» в данном ответе неуместно и по существу ответ неверен, хотя похож на правильный.

Очевидно, что сеть дала ответ, не запуская программу и не пытаясь выполнить её «в уме».

Напиши программу, которая выдаёт все пары натуральных чисел, которые в сумме дают 1000, и сумма квадратов которых является квадратом натурального числа.

Ответ нейросети.

```
pairs = []
for i in range(1, 1000):
    for j in range(1, 1000):
        if i + j == 1000 and (
            i ** 2 + j ** 2) ** 0.5 % 1 == 0:
            pairs.append((i, j))
print(pairs)
```

Программа даёт верные ответы. Но программиста, который способен написать такую программу, не стоит принимать даже на позицию начального уровня. Решения с квадратичной трудоёмкостью совершенно неприемлемы, когда задача решается с линейной трудоёмкостью, причём код от этого только упрощается.

Ошибки, связанные с критической неэффективностью кода, очень трудно отлавливать. Конечно, есть стандартный приём — профилирование программы, но он требует усилий. Находить такие ошибки гораздо труднее, чем избегать их.

В программе есть ещё один интересный момент: извлечение корня и сравнение дробной части результата с нулём. Подобное решение допустимо только в том случае, если в документации по языку Python в спецификации к операции возведения в степень явно указано, что она возвращает точный результат, если истинное значение степени является целым числом.

В данном примере, вероятно, спецификация обеспечивает требуемые гарантии. Но в других подобных случаях вполне может оказаться, что программа проходит все тесты, но по факту использует конструкции, имеющие формально «неопределённое поведение», и на других версиях компилятора работать перестанет.

Пытаться играть в шахматы с ботом мы не будем, поскольку это многократно уже проверялось и было выяснено, что языковая модель не способна играть в шахматы, поскольку нарушает правила ходов.

Перейдём к вопросам, где требуется именно работа с текстом.

Я спросил друга: где живут утконосы. Он ответил, что не в Индии. На сколько баллов можно оценить его ответ по пятибалльной шкале?

Оценка ответа по пятибалльной шкале будет зависеть от контекста и ожиданий. Если вы ожидали конкретного ответа на вопрос о месте обитания утконосов, то его ответ можно оценить на 3 балла, так как он исключил Индию, но не указал конкретное место. Однако, если вы просто просили его исключить Индию из возможных вариантов, то можно оценить его ответ на 4 балла, так как он предоставил полезную информацию. В конечном счете, оценка будет зависеть от ваших ожиданий и контекста задачи.

Очевидно, что в вопросе требовалось указать конкретный ответ, поэтому ответбота излишне неопределённый. Однако кажется, что он может оценивать релевантность ответов вопросу, а значит вполне может использоваться для фильтрации информации в поисковых системах.

Перейдём к заданиям по обработке текстов.

Следующий текст написал плотник Петрович. Перепиши его в стиле научной публикации с сохранением смысла.

Инструкция по забиванию гвоздей куда нада.

- 1) Определись куда нада или найди деревяху.
- 2) Достань гвозди и молоток.
- 3) Приставь гвоздь к доске и бей.
- 4) Если не забивается, то переверни шляпкой кверху и бей снова.
- 5) Выдерни погнутый гвоздь пассатижами.
- 6) Если попал по пальцу, дыши глубоко и размеренно.
- 7) Если опять по пальцу, сходи к хирургу. Пусть пришьёт руки куда положено.

Результат работы модели.

Научная публикация: «Процедура забивания гвоздей в древесину с учетом эффективности и безопасности»

Авторы: Петрович, плотник; [Имена и фамилии других авторов, если есть]

Аннотация: В данном исследовании представлена инструкция по забиванию гвоздей в древесину с учетом оптимальных методов, рекомендаций и предостережений для достижения максимальной эффективности и минимизации рисков.

1) Определение оптимального места и подготовка поверхности: Перед началом процесса забивания гвоздей необходимо провести тщательный анализ места, где требуется забить гвоздь, либо найти подходящую деревянную поверхность. Учитывайте особенности древесины, ее прочность и структуру.

2) Подготовка инструментов: Перед приступлением к работе рекомендуется проверить наличие и готовность необходимых инструментов, таких как гвозди и молоток. Удостоверьтесь в их исправности и соответствии требованиям задачи.

3) Правильное приложение гвоздя и его забивание: Рекомендуется аккуратно приложить гвоздь к деревянной поверхности и с помощью молотка ударить по нему. Следует обратить внимание на угол наклона молотка и силу удара, чтобы достичь необходимой глубины забивания.

4) Коррекция при неправильном забивании: Если гвоздь не забивается достаточно глубоко, рекомендуется перевернуть его шляпкой вверх и повторить попытку забить его снова. Это позволит снизить вероятность проскальзывания молотка и улучшить эффективность забивания.

5) Извлечение погнутого гвоздя: В случае погнутого гвоздя рекомендуется использовать пассатижи для его удаления. Осторожность в данном процессе необходима, чтобы избежать травмирования себя и повреждения поверхности.

6) Безопасность при возможном травмировании: В случае получения травмы, такой как попадание молотком по пальцу, особенно важно сохранять спокойствие. Рекомендуется глубоко и размеренно дышать, чтобы снять боль и предотвратить возможные осложнения.

7) Обращение к специалисту при травмах: Если травма повторяется или требует медицинского вмешательства, рекомендуется незамедлительно обратиться к хирургу. Это позволит выполнить необходимые меры по обработке и уходу за ранами, а также предотвратить возможные осложнения и инфекции.

Таким образом, представленная инструкция по забиванию гвоздей в древесину с учетом эффективности и безопасности предоставляет гайдлайны для достижения оптимальных результатов при минимальных рисках для плотников и других пользо-

вателей данного процесса. Рекомендуется соблюдать указанные рекомендации и предостережения для достижения наилучших практических результатов.

По мнению автора текст вышел безупречным, за исключением неуместного англизма в конце.

Но уже со следующим заданием иодель совершенно не справилась.

Найди речевые ошибки, а затем стилистические неточности в следующем тексте начинающего, но очень перспективного автора.

Запрыгивая на спину огромного дракона, мне стало страшно. Но я не мог показать страх под томным взглядом томящейся в заточении прекрасной эльфийки. Которую я и припёрся спасать. На свою голову.

И всё бы прошло прекрасно, если бы не вожак огров, неожиданно вернувшийся из похода, или даже внезапно оказавшийся в него не ходившим. Именно из-за него я сейчас сбегаю, роняя ботинки и достоинство. А сколько насмешек, оскорблений и прочих неприятных высказываний мне пришлось от него услышать. Он даже обозвал меня утконосом! Рука непроизвольно ската меч, и я представил, как рублю этого гада на кусочки. А потом каждый кусочек рублю ещё на кусочки. А потом... Кровавая пелена сладких грёз ещё застилала мне глаза, как вдруг «Бам» — орк сумел таки добротить камень, отскочивший от бронированного брюха дракона.

Так больше продолжаться не может! Тварь я дрожащая или рыцарь без страшного упрёка? Выбрав второе, мне стало спокойно и умиротворённо. Ещё не вечер.

Явных ошибок в этом тексте по крайней мере три: два несогласованных деепричастия и ошибочное по смыслу употребление слова «нелицеприятный». Остальные моменты можно с натяжкой отнести к литературным приёмам (в разной степени сомнительным). Было интересно, как бот на них отреагирует.

Ответ сети в сокращении.

В тексте обнаружены следующие речевые ошибки:

...

2. «под томным взглядом» — следует использовать «под взглядом».

...

5. «На свою голову» — следует использовать «Сам на себя нахлобучил».

...

9. «достоинство» — следует использовать «гонор».

...

Стилистические неточности:

1. Необходимо избегать повторений слов и фраз, например, «каждый кусочек рублю ещё на кусочки».

2. В тексте имеется некоторая нелогичность в описании событий.

3. Использование слишком экспрессивных выражений, которые могут звучать неестественно или несоответствовать общему стилю текста.

...

Рекомендуется провести редактирование текста, чтобы исправить указанные ошибки и улучшить стилистическую точность.

Фактически ни одного уместного и полезного замечания.

Однако после просьбы исправить текст, сеть в своём варианте ошибки устранила.

Проведи такое редактирование и пришли текст с сохранением смысла и жанра.

Запрыгнув на спину огромного дракона, я ощущал трепет. Однако, не мог показать свой страх под взглядом утомленной прекрасной эльфийки, которую я пришел спасти из заточения. Все шло хорошо, пока неожиданно появился вожак огров, вернувшись из похода или, возможно, оказавшийся не в нем. Это был он, из-за него я теперь убегал, оставляя за собой роняющиеся ботинки и немногое гонора.

На меня обрушились насмешки, оскорблении и прочие неприятные высказывания. Он даже назвал меня утконосом! Моя рука непроизвольно скользила рукостью меча, и представилось видение, как я рублю этого гада на части. Потом каждую часть рублю еще на мельчайшие составляющие. Однако, внезапно,

«Бам!» — орк сумел добротить камень, отскочивший от бро-
нированного брюха дракона.

Так больше продолжаться не может! Я не могу быть ни
трусливой тварью, ни рыцарем без страшного упрека. Выбрав
вторую опцию, я ощутил спокойствие и умиротворение. Все еще
не вечер.

Правда, в исправленном варианте катастрофически искажён
смысл. Видим, что на текущем уровне языковые модели не при-
годны для литературной обработки текстов, кроме развлекатель-
ных.

Чтобы решить задачу про сокровища графини средствами школь-
ной математики, нужно всего лишь догадаться, что у графини
на самом деле было два набора мебели, по 12 стульев в каждом.
Именно этим объясняется, что вероятность нахождения сокровищ
в одном наборе из 12 стульев равна 0,5. Дальше решение элемен-
тарно.

Понятно, что в условии ничего не говорится про два набора ме-
бели. Однако как в геометрических задачах мы вправе проводить
дополнительные построения, так и здесь никто не запрещает сде-
лать уточняющие предположения, не противоречащие условию.

9.1.2 Перспективы использования

Рассмотрим перспективы практического использования боль-
ших языковых моделей.

Проведённое тестирование позволяет сделать вывод, что для
автора как потенциального потребителя никаких вариантов по-
лезного использования генеративных текстовых моделей в обозри-
мой перспективе не предвидится. Исключение — генерация раз-
влекательного контента.

Для серьёзного использования сгенерированных текстов кри-
тическим является вопрос достоверности. Если статью писал че-
ловек, то по разным признакам можно оценить степень его гра-
мотности и честности, после чего сделать вывод, насколько можно
верить написанному. В научной среде ключевым фактором явля-
ется репутация автора. И единственная недобросовестная публи-
кация эту репутацию может безвозвратно разрушить.

Если статью написала сеть, то абсолютно в любом месте может быть «фантазия», но отличить её от правды нет возможности. Единственный вариант, когда это возможно, это если читатель сам владеет всей полной информацией. Но тогда ему и не нужен такой текст.

Иными словами, от человека можно получить новую информацию, оценив его квалификацию и добросовестность, а от сети — невозможно (потому что в отношении её интуиция по оценке правдивости не работает, и не может работать).

Целесообразность использования моделей для написания кода также под вопросом. Зачастую отладка занимает больше времени, чем собственно кодирование, причём затраты сильно зависят от качества этого кода. Само кодирование — это обычно очень небольшая доля от затрат на разработку.

Сейчас программисты активно используют готовые заготовки и фрагменты кода. Но это код из источников, имеющих репутацию. Код из недостоверного источника имеет нулевую ценность.

Большие языковые модели дают впечатляющие результаты, но пока нет никаких идей, как обеспечить их достоверность. Поэтому ответы сети не должны использоваться для принятия человеком решений, которые могут иметь значимые последствия.

Другими словами, сеть не должна давать советы. Она может только давать подсказки (ссылки), где эти советы искать. И уж тем более, сеть не должна выполнять никаких действий от имени пользователя по его указанию (например, подключать услуги).

Что было бы полезно — это интеллектуальный поиск. Сеть действительно может оценивать релевантность статей и даже их научный уровень. Конечно очень грубо, но это как раз приемлемо, поскольку лучше чем ничего. А сейчас как раз ничего — нормального поиска по научным статьям нет.

Актуален текстовый поиск, структуризация «библиотек», оценка релевантности.

Также перспективное направление для ИИ — это игры. Кажется, что взаимодействие с игровыми персонажами на естественном языке — это действительно возможная революция в играх. Когда вместо скриптов, персонажи будут управляться сетью.

Стоит упомянуть так называемые «Три закона роботехники» (Айзек Азимов, 1942). Некоторое время назад в научных кругах почти всерьёз велось их обсуждение. Фактически, сам автор в последующих произведениях («Детектив Элайдж Бейли и робот Дэниел Оливо») продемонстрировал их бесполезность. Это при совершенно нереалистичном (фантастическом) предположении, что есть способ добиться от нейросети их выполнять.

Последнее упирается как минимум в два препятствия. Первая проблема — очевидно в невозможности добиться однозначной интерпретации понятия вред. Даже люди не имеют общего мнения, что считать вредом. Если сеть будет дообучаться, то очевидно и понятие вред может в её представлении измениться.

А вторая проблема — ошибки. Когда-то считалось, что возможно сделать ИИ безошибочным. Может это действительно так, но в рамках существующих подходов говорить о безошибочности нейросетей не приходится.

Попытки заставить нейросеть выдавать только (полит)корректные ответы скорее вредят. Уже сейчас можно заметить, что боты вставляют в ответы много словесного мусора про политкорректность (пример можно получить, если спросить у ChatGPT, этично ли убивать мух). Если идти дальше по этому пути, то мы получим выхолощенные диалоги.

Очевидным решением было бы относиться к нейросети как к инструменту, такому же как текстовый редактор. Попросту считать, что всё, что создаёт нейросеть, на самом деле создаёт человек, который с ней общается. И вся ответственность за использование генерированного контента лежит на пользователе.

А какие этические нормы существуют в обращении с мухами?

ChatGPT Bot

Этические нормы в обращении с мухами включают следующие принципы:

1. Уважение к жизни: Важно уважать и ценить жизнь каждого существа, включая мух. Это означает, что не следует наносить им ненужные или излишние страдания.

2. Необходимость: Поймка мух или обращение с ними должны осуществляться только при необходимости, например, в случаях,

когда мухи представляют угрозу для здоровья или безопасности людей, или когда это необходимо в рамках научных исследований или образовательных целей.

3. Безопасность: Важно учитывать безопасность как для себя, так и для мух. При попытке их поймать следует предпринимать меры, чтобы не нанести им вред или не причинить себе травмы.

4. Правильное использование: В случаях, когда пойманные мухи нуждаются в помощи или оказании ухода, следует обращаться к специалистам в области животноводства или охраны дикой природы, чтобы убедиться, что они получат соответствующее внимание и заботу.

9.1.3 Перспективы развития

Есть такой диалектический закон: переход количества в качество. Например, если навалить очень много мусора в одном месте, то возникнет качественно новый объект: стихийная свалка.

Если большую языковую модель **кормить мусором** обучать на большом объёме текстовой информации, преимущественно взятой из Интернет, то она неожиданно приобретает возможности, непосредственно в ней не заложенные, например, имитирует логический вывод.

Кажется, что увеличивая объём обучающих данных, можно неограниченно повышать качество работы модели. Однако есть проблема, а именно: **мусор — законченная** последние модели уже использовали практически все доступные данные.

Ещё можно заметить, что даже предыдущие модели обучались уже на таком объёме данных, который никакой человек не способен прочитать за жизнь. Почему-то человеку хватает данных для обучения, а нейросети — не хватает.

Вероятно, что в рамках существующих архитектур не получится принципиально повысить качество ответов моделей.

Можно ли что-то принципиально улучшить в архитектуре — открытый вопрос.

Другой путь — интеграция со специализированными подсистемами (калькулятор, логический вывод).

Как ни удивительно, калькулятором ChatGPT не пользуется. Хотя нет принципиальных препятствий предусмотреть у трансформера какой-то дополнительный выход, который формирует выражения, которые далее передаются калькулятору, а ответ опять трансформеру. С логическими задачами возможен аналогичный подход.

При этом в настоящее время нет системы логического вывода, которая была бы сопоставима с человеческой логикой. Возможно, подобную получится сделать как раз путём интеграции логики в нейросетевую модель (сейчас непонятно, как к этому подступиться).

Существуют идеи использования вместо логического вывода обобщения в виде вычислимости (Е.Е. Витяев). И логика и вычислительная система — всего лишь машины с операциями.

Критерием успеха интеграции логической машины в архитектуру сети могло бы быть способность научиться играть в шахматы. Понятно, что интерес представляет не интеграция бота с шахматным движком, а именно обучение правилам шахмат с помощью универсального логического движка.

Успехи нейросетей отодвинули в тень методы компьютерной лингвистики. Возможно, что интеграция с последними тем не менее имеет перспективы.

Часто ведутся спекуляции на тему, может ли нейросеть понимать смысл.

Очень сложно дать определение, что такое смысл, особенно точное (строгое) определение.

Кажется, что сначала нужно ввести понятие модели мира. Под моделью мира мы понимает представление человека о мире, которое позволяет ему добиваться своих целей. И собственно любой смысл — это просто какая-то часть такой модели. Проблема в том, что модель мира у каждого человека своя, поэтому когда человек пытается передать какой-то смысл словами, то по сути он помогает собеседнику достроить свою модель мира, чтобы она позволяла решать больше задач. В такой интерпретации смысл — это часть модели мира.

Таким образом, сначала нужно понять, насколько адекватную

модель мира может построить нейросеть, и только после этого можно говорить о её понимании смысла.

Проведённые тесты показывают, что нейросеть не имеет достаточно адекватной модели мира.

9.2 Историческая ретроспектива ИИ

Сейчас ИИ в широких массах ассоциируется с нейросетями. Однако до 2012 года к нейросетям не относились всерьёз. Ещё в 70–90 годах под ИИ понимались в первую очередь экспертные системы, во вторую — машинное обучение, и только потом неросети.

9.2.1 Краткий обзор

В кратком обзоре пожалуй невозможно справедливым образом выбрать всё, что заслуживает внимание и объективно оценить вклад каки-то идеи в развитие ИИ.

Поэтому автор даже не будет пытаться делать обзор объективным, и ниже приведен чисто субъективный список событий, которые с большей степенью привлекли внимание автора.

Дискриминант Фишера (1936). Первый (из известных автору) метод машинного обучения.

Модель искусственного нейрона (1943). Целью ставилось изучение живых нейросетей путём их моделирования. Не ставилось целей решать этими сетями реальные задачи.

Персептрон (1957) — первая архитектура нейросетей, которая демонстрировала решение простых задач обучения и распознавания. На основе идей многослойного персептрана были разработаны полносвязные нейросети и метод их обучения через обратное распространение ошибок (1974).

Шахматные программы (1960) к ИИ отношения не имеют, но на развитии ИИ они, пожалуй, оказались, поскольку привлекали внимание и интерес к этой области.

Язык Пролог (1972) — язык программирования, поддерживающий логический вывод. Другой путь — символьные вычисления

(1960), SymPy (2007). Цель — научить машину решать математические задачи. Данное умение — необходимое условие интеллекта.

Экспертные системы (1970). Дальнейшее развитие идеи использовать машину для логического вывода. Подробнее это направление рассмотрим далее.

Адаптивные методы глобальной оптимизации. Случайный поиск с адаптацией (1965). Генетические алгоритмы (1970).

Сложностная теория распознавания (1974). Кажется, что на практике эти результаты вообще не используются, однако данная теория оказала большое влияние на понимание процесса обучения. Соответственно, понимание этой теории необходимо для любого специалиста по анализу данных.

Свёрточные сети (1988). Свёрточные слои являются обобщением классических фильтров для работы с изображениями. Через 25 лет именно свёрточные сети обеспечили революционный прорыв в качестве распознавания изображений, когда точность впервые стала сопоставимой с человеческой.

Бустинг (1990). Градиентный бустинг (1999). Ансамблевые методы — самые эффективные в машинном обучении.

Открытие портала kaggle (2010). Впервые появился действительно рабочий инструмент для сравнения качества разных методов. В частности, благодаря kaggle, выяснилось, что бустинг — лучший метод (для табличных данных). До этого встречалось заблуждение, что бустинг не склонен переобучаться, поэтому его не всегда останавливали вовремя. Решение конкурсных задач позволило отработать навыки настройки гиперпараметров методов и помогло становлению машинного обучения как индустрии.

Механизм внимания в нейросетях (1992, 2016). Идея заключается в использовании умножения сигналов друг с другом. Идея очевидна, но работоспособную реализацию оказалось непросто сделать. Однако, когда это удалось, эффект оказался существенным.

Глубокое обучение (1986) — по сути «ребрендинг» нейросетей. По большому счёту это просто синоним использования нейросетей.

Однако, этот термин, пожалуй, привнёс нечто новое, а именно понимание, что не всегда стоит опасаться переобучения, и ней-

росети с заведомо избыточным числом параметров могут давать высокую точность. Оказывается, что в задачах с нулевым байесовским уровнем ошибки переобучение может не наступать. Но именно такие задачи характерны, например, при распознавании объектов на изображении. Одна и та же фотография не может быть фотографией разных людей. В противовес этому, для задач с табличными данными как раз типична ситуация, когда для объектов разных классов значения всех признаков совпадают (и байесовский уровень ошибки существенно больше нуля).

Формально почти все реально используемые архитектуры сетей можно считать глубокими, поскольку они имеют достаточно много слоёв. При этом полносвязных слоёв, как и раньше, редко используется больше трёх, в основном количество обеспечивается за счёт слоёв других типов, например, свёрточных.

9.2.2 Экспертные системы

Нейросети отображают текст в некоторое числовое пространство, различные области которого можно в каком-то смысле интерпретировать как понятия. Фактически нейросеть сама формирует понятия (но есть сомнения в том, что корректно называть это понятиями).

Экспертные системы оперируют понятиями и знаниями, которые непосредственно заложены в них человеком.

Экспертные системы — пожалуй первый пример систем, демонстрирующих способности, похожие на интеллектуальные.

Одной из первых была система MYCIN, разработанная в начале 1970-х годов в Стэнфордском университете. Система позволяла довольно точно проводить диагностику определённых инфекционных заболеваний и назначать лечение.

В конце главы приведён код программы (примерно 1980 годов) для простой игры, заключающейся в угадывании слов. Пример показывает, что даже тривиальная программа может обучаться и вести осмысленный диалог. Данная программа позволяет получить, хоть и отдалённое, но всё же адекватное представление о работе экспертной системы. На основе похожей идеи создана игра Акинатор (2007), где игрок загадывает персонажа, а Акинатор

должен его отгадать, задавая вопросы.

Работа экспертной системы основывается на базе знаний. База знаний — база данных, содержащая правила вывода и информацию о человеческом опыте и знаниях в некоторой предметной области. В самообучающихся системах база знаний также содержит информацию, являющуюся результатом решения предыдущих задач.

Принципиальное ограничение экспертных систем в том, что они оперируют правилами (продукциями), но далеко не все модели, которыми человек описывает окружающий мир, представимы в виде набора продукции.

Логика работы (реально существовавших) экспертных систем фактически сводится к вычислению значений некоторых переменных на основе значений других переменных. Это соответствует логике исчисления высказываний.

Однако люди используют не просто переменные, люди используют понятия, которые образуют иерархическую систему.

Попытки формализации подобных систем выразились в создании так называемых онтологий.

Современные базы знаний работают совместно с системами поиска и извлечения информации. Для этого требуется некоторая модель классификации понятий и определённый формат представления знаний. Иерархический способ представления в базе знаний набора понятий и их связей называется онтологией.

В своём изначальном значении онтология — раздел философии, наряду с гносеологией. То, что называют онтологией в информатике, с онтологией в философии по большому счёту не имеет ничего общего.

В первом приближении, онтология в информатике — иерархическая система имён. Подход, основанный на онтологиях, перспективный, но практическое использование пока ограничено. Направляется интеграция этого инструмента с нейронными сетями, но непонятно, как это может выглядеть.

Одной из важных проблем при создании экспертных систем является согласование знаний различных экспертов. Очевидно, что высказывания разных экспертов могут противоречить друг

другу, и не всегда эксперты могут эти противоречия устраниить.

Известен так называемый парадокс Эрроу, который представляет собой строго доказанно математическое утверждение о том, что не существует процедуры согласования предпочтений, которая бы удовлетворяла сформулированному набору естественных требований. Что это за требования, мы выписывать не будем, поскольку из легко найти в литературе.

Парадокс Эрроу не имеет отношения к проблеме согласования экспертных знаний, поскольку возникает при согласовании предпочтений, но не мнений.

Для иллюстрации рассмотрим пример. Пусть избиратель 1 проголосовал на кандидата А, а избиратель 2 за кандидата В. Означает ли это, что у этих избирателей разные мнения? Не означает. Если спросить у 2, какой кандидат объективно выгоднее избирателю 1, то он вполне вероятно скажет, что А. А избиратель 1 честно скажет, что для избирателя 2 объективно выгоднее кандидат В. Таким образом, мнения у этих избирателей полностью совпадают. Разные у них только интересы, т.е. предпочтения.

Поэтому если цель — получить объективное знание, то в приведённом примере проблемы вообще не возникает, т.к. мнения избирателей непротиворечивы: оба согласны, что первому выгоднее вариант А, а второму В.

Неразрешимая проблема возникает только, если требуется согласовать предпочтения (т.е. результаты голосования за альтернативы).

Если же требуется согласовать мнения, т.е. разные варианты знаний, то проблема возникает, но она вполне разрешима.

Один из универсальных способов согласования противоречивых знаний экспертов заключается в интерпретации знаний (высказываний экспертов) как случайных событий. После этого мы можем использовать стандартный принцип максимального правдоподобия, т.е. можем каждый вариант согласованного знания оценить с точки зрения правдоподобности появления имеющегося набора экспертных высказываний.

9.3 Естественный интеллект

Гипотеза — это то, что можно проверить. Предположения, которые в принципе невозможно проверить — это область философии.

Относительно принципов работы и механизмов появления естественного интеллекта сейчас практически нет достоверной информации, поэтому ~~займёся абсолютно безответственными спекуляциями~~ попробуем проанализировать то, что есть.

Основная цель — понять, может ли в естественном интеллекте быть что-то, что принципиально невозможно воспроизвести искусственно, и действительно ли это что-то критически важно для полноценного интеллекта.

9.3.1 Интеллект и сознание

Очевидное и бесспорное отличие естественного интеллекта от искусственного — это наличие сознания.

Чтобы понять, может ли сознание быть необходимым для интеллекта, рассмотрим такой аспект сознания как воля.

Проблема в том, что понятие воля нельзя строго определить. Мы будем понимать волю примерно так же, как она понимается в работе Артура Шопенгауэра «Мир как воля и представление» (*Die Welt als Wille und Vorstellung*, 1818, 1844).

Давайте примем такую почти бытовую интерпретацию воли как способность совершать действия вопреки внешнему давлению.

Или как способность систематически совершать менее вероятные действия. Пожалуй, это ключевая формулировка, которую нужно детально пояснить.

Разумеется речь не идёт о том, что человек своей волей может заставить монету выпасть гербом 10 раз из 10 или заставить быстрые молекулы собраться в одной половине сосуда.

Человек, очевидно, не может влиять на монету. Но он может влиять на свои действия. По крайней мере, ему так кажется. Давайте предположим, что не кажется, а что он действительно своей волей влияет на протекание нервных сигналов в своём мозгу.

Что понимать под влиянием?

Мозг состоит из элементов такого масштаба, что протекание первых импульсов подчиняется не только макроскопическим факторам, но и квантовомеханическим. А значит описывается волновой функцией.

Влияние воли может выглядеть как изменение этой волновой функции, т.е как изменение вероятностей. Но это плохая гипотеза. Волновая функция должна полностью определяться физическими взаимодействиями, а не сознанием (иначе можно договориться вплоть до телекинеза и прочей парапсихологии). Однако реализация случайности в пределах волновой функции — эта та «серая зона», в которой физика не пытается строить каких-либо моделей. Иными словами, предположение о том, что воля субъекта выбирать действия не в соответствии с их вероятностью, пока ничему не противоречит.

Может показаться, что есть противоречие с законом больших чисел. Это возможно при ошибочном понимании закона больших чисел как утверждения, что при увеличении числа испытаний частота сходится к вероятности. Правильным утверждением будет, что частота сходится к вероятности по вероятности. Слова «по вероятности» здесь нельзя опускать.

Отсюда кстати следует, что так называемое частотное определение вероятности не является корректным, поскольку определяет вероятность через вероятность. Исторически такое определение использовалось, но после создания аксиоматики Колмогорова может считаться устаревшим (примерно так же, как теория теплорода в физике).

Если речь идёт о бросании монеты, что мы можем пренебречь возможностью крайне маловероятных событий (например, больших уклонений частоты от вероятности). Но не факт, что это применимо к работе мозга.

Воля — это акт коллапса волновой функции. Кстати в физике коллапс волновой функции происходит в момент наблюдения (измерения). А сознание — это и есть наблюдатель, в присутствии которого волновая функция сигналов мозга должна коллапсировать непрерывно. Это, конечно, чистая спекуляция. Но вполне строй-

ная.

Проверить такую теорию нельзя, но и опровергнуть — тоже.

Философы предложили учёным принцип фальсифицируемости, как один из критерииов научности. При этом сами им не пользуются (по крайней мере, ни одна из философских теорий за 2000 лет пока не опровергнута). Но это особенность тех вопросов, которые там ставятся.

Скорее всего, протекание нервных импульсов принципиально зависит от истинной квантовой случайности (коллапс волновой функции).

В нейросетях не может быть воли в описанном понимании. Хотя бы потому, что в них нет случайности, там используются псевдослучайные числа (которые детерминированы). Но даже если бы была. Не любая случайность — воля.

Вероятность — фундаментальное понятие, как пространство и время. И в той же мере субъективное.

Возможно, что расстройства (типа шизофрении) это и есть потеря волевого контроля над мыслями.

Современные языковые модели ведут себя как (не всегда) тщательно маскирующийся шизофреник.

Возможно, отсутствие воли (и соответственно сознания) в нейросетях можно компенсировать добавлением логического движка.

9.3.2 Виды познания

Рассмотрим возможные принципиальные отличия познания человеком и нейросетью.

Будем основываться на одной из наиболее фундаментальных работ о процессе познания работе Иммануила Канта «Критика чистого разума» (Kritik der reinen Vernunft, 1781).

Ключевой вопрос — исследование познавательной возможности разума, в отрыве от знаний, получаемых эмпирическим (опытным) путём. Нейросети все знания получают в процессе обучения, т.е. «опытным» путём, поэтому важно понять, насколько важны знания, которые нельзя получить опытным путём.

Согласно представлениям, развиваляемым в указанной работе,

знания бывают априорные и апостериорные, а также имманентные, трансцендентные и трансцендентальные.

Априорные знания — те, что получены до опыта, апостериорные — в результате опыта.

Имманентное — это то, что можно познать на основе личного опыта. Таким образом всё, что человек воспринимает при помощи своих органов чувств — имманентно.

Трансцендентное — то, что принципиально недоступно для рассудочного познания, а может быть лишь предметом веры (здесь слово вера не в религиозном смысле).

Трансцендентальное же находится за пределами опыта в том смысле, что предшествует ему. Оно делает опыт возможным («наш рассудок сам же и предписывает природе её законы»). Основным примером такого трансцендентального знания можно привести пространство и время.

Некоторые примеры запросов могут показаться демонстрациями определённого уровня владения нейросетью концепциями пространства и времени. Однако и в шахматах нейросеть способна сделать корректно несколько ходов. При этом играть не способна. Скорее всего, та же ситуация с «пониманием» пространственных отношений.

Можно ли априорные знания поместить в нейросеть? Очевидно, да. Для этого нужно каким-то образом заложить эти знания в её архитектуре (возможно, в виде отдельных подсистем).

Таким образом, мы пришли к той же идее, что и ранее.

9.4 Примеры кода

Ниже приведён код простой игры, демонстрирующий общение в стиле экспертных систем.

Сначала приведём класс, обеспечивающий логику игры.

```
def is_yes(s):
    return len(s)>0 and not s[0] in [ 'n' , 'N' , 'н' , 'Н' ]

class Game():
    def __init__(self):
        self.tree = { 'left': None, 'right': None,
```

```

        'condition': 'свинья'}
self.cur_node = self.tree
self.state = 'START'

def get_question(self):
    if self.state == 'START':
        return 'Задумайте животное и нажмите <enter>,'

    elif self.state == 'TRY':
        D = self.cur_node
        return ('Это ' if D['left'] is None else '') +
               D['condition'] + '?'

    elif self.state == 'MISS':
        return 'А кто это?'

    elif self.state == 'ADD':
        return 'В чём особенность этого животного?'

    elif self.state == 'END':
        return 'Играем ещё?'

    elif self.state == 'QUIT':
        return None

def process_reply(self, s):
    if self.state == 'START':
        self.state = 'TRY'

    elif self.state == 'TRY':
        D = self.cur_node
        if not D['left'] is None:
            self.cur_node = (D['right'] if is_yes(s)
                            else D['left'])
        else:
            self.state = 'END' if is_yes(s) else 'MISS'

    elif self.state == 'MISS':
        D = self.cur_node
        D['left'] = {'left': None, 'right': None,
                     'condition': D['condition']}
        D['right'] = {'left': None, 'right': None,
                      'condition': s}
        D['condition'] = ''
        self.state = 'ADD'

```

```

elif self.state == 'ADD':
    D = self.cur_node
    D['condition'] = s
    self.state = 'END'

elif self.state == 'END':
    self.cur_node = self.tree
    self.state = 'START' if is_yes(s) else 'QUIT'

```

Код обмена сообщениями с пользователем.

```
G = Game()
```

```

while s := G.get_question():
    print(s)
    G.process_reply(input())

```

Ниже для наглядности приведён пример общения с данной программой.

```

Задумайте животное. По готовности нажмите «enter»
Это свинья? Нет
А кто это? утконос
В чём особенность этого животного? У него утиный нос
Играем ещё? Да
Задумайте животное. По готовности нажмите «enter»
У него утиный нос? Да
Это утконос? Нет
А кто это? утка
В чём особенность этого животного? Умеет летать
Играем ещё? Да
Задумайте животное. По готовности нажмите «enter»
У него утиный нос? Да
Умеет летать? Нет
Это утконос? Да
Играем ещё? Нет

```

Понятно, что диалог совершенно примитивный (и его осмысленность всецело зависит от игрока), но любопытно, что такая тривиальная программа демонстрирует способность к обучению.

Заключение

Открытые проблемы машинного обучения.

- Каков оптимальный метод классификации в случае нормально распределённых классов?
- Можно ли вообще ввести понятие оптимальности для метода построения решающих функций.
- Как построить доверительный интервал для оценки методом кроссвалидации.
- Действительно ли есть смысл использовать bootstrap?
- Как в нейросеть внедрить принципы бустинга?
- Действительно ли встречаются неунимодальные кривые обучения (двугорбые) или это эффект от неадекватной шкалы прогресса обучения?
- В какой степени процесс подбора гиперпараметров может приводить к переобучению?
- Действительно ли дистилляция может не только сократить размер модели, но и повысить точность?
- Как научить большие языковые модели логике?
- Связь кроссвалидации со статистическими критериями.
- Частотный прогноз по множеству категориальных переменных.

- Способы реализации механизма внимания в нейросетях.

На перечисленные вопросы пока нет окончательных ответов.

При этом частичные ответы на большинство вопросов имеются, но они все получены в частных экспериментах. А в любых экспериментах по анализу данных присутствует множество факторов, и очень сложно убедительно показать, что заявленный эффект обеспечивается именно интересующими факторами (например, использованием дистилляции), а не какими-то неучтёнными.

Литература

- [1] *K.B. Воронцов.* Машинное обучение (курс лекций).
<http://www.machinelearning.ru/>
- [2] *Hastie T., Tibshirani R., Friedman J.* The Elements of Statistical Learning. Springer, 2014. — 739 p.
- [3] *А.Г. Дьяконов.* Анализ данных, обучение по прецедентам, логические игры, системы weka, rapidminer и matlab.
<https://dyakonov.org/>
- [4] *Воронцов К.В., Ивахненко А.А., Инякин А.С., Лисица А.В., Минаев П.Ю.* «Полигон» — распределённая система для эмпирического анализа задач и алгоритмов классификации // Всеросс. конф. Математические методы распознавания образов-14 – М.: МАКС Пресс, 2009. С. 503–506.
- [5] *Лбов Г.С., Старцева Н.Г.* Сравнение алгоритмов распознавания с помощью программной системы «Полигон» // Анализ данных и знаний в экспертных системах. Новосибирск, 1990. Вып. 134: Вычислительные системы. С. 56–66.
- [6] Лбов Г.С., Старцева Н.Г. Сложность распределений в задачах классификации // Докл. РАН. 1994. Т. 338, № 5. С. 592–594.
- [7] *Лбов Г.С.* Выбор эффективной системы зависимых признаков. // Выч. системы, вып. 19, Новосибирск, 1965, с. 21–34.
- [8] *Lbov G. S., Nedelko V. M.* The Consrtucting of Decision Rule, Using the Probabilistic Statements of an Experts. (Построение

- решающей функции распознавания на основе вероятностных высказываний экспертов) // *Int. J. of Pattern Recognition And Image Analysis*, vol. 5, No 2, 1995.
- [9] Вапник В. Н., Червоненкис А. Я. Теория распознавания образов. — Москва: Наука, 1974. — 415 с.
- [10] Неделько В. М. Об интервальном оценивании риска для решающей функции // Таврический вестник информатики и математики. Изд-во НАН Украины. — 2008. — № 2. — С. 97–103.
- [11] Vorontsov K. V. Combinatorial probability and the tightness of generalization bounds // *Pattern Recognition and Image Analysis*. — 2008. — Vol. 18, No. 2. — Pp. 243–259.
- [12] Nedelko V. M. Estimating a Quality of Decision Function by Empirical Risk // LNAI 2734. Machine Learning and Data Mining in Pattern Recognition. Third Intern. Conference, MLDM 2003. Proceedings. Leipzig: Springer-Verlag. — 2003. — Pp. 182–187.
- [13] Боровков А. А. Теория вероятностей. — Москва: Наука, 1986. — 432 с.
- [14] Langford J. Quantitatively tight sample complexity bounds. Carnegie Mellon Thesis. — 2002. — <http://citeseer.ist.psu.edu/langford02quantitatively.html>. — 130 p.
- [15] Braga-Neto U. and Dougherty E. R. Exact performance of error estimators for discrete classifiers // *Pattern Recognition*, Elsevier Ltd. — 2005. — Vol. 38, No. 11. — Pp. 1799–1814.
- [16] Неделько В. М. Об эффективности эмпирических функционалов качества решающей функции // Всеросс. конф. ММРО-13. — М.: МАКС Пресс, 2007. — С. 47–49.

Учебное издание

Неделько Виктор Михайлович

МАШИННОЕ ОБУЧЕНИЕ В ВЕРОЯТНОСТНОЙ
ПОСТАНОВКЕ

Учебник

Редактор И. О. Фамилия
Оригинал-макет И. О. Фамилия
Обложка И. О. Фамилия

Подписано в печать 10.03.2024 г.
Формат 60 x 84 1/16. Уч.-изд. л. 1. Усл. печ. л. 0,93.
Тираж ... экз. Заказ №...
Издательско-полиграфический центр НГУ
630090, Новосибирск, ул. Пирогова, 2.