

# Математика и алгоритмы машинного обучения

Глинский А.В.

# Содержание

<b>1 Основные термины в машинном обучении. Базовая математика в машинном обучении</b>	<b>9</b>
1.1 Вводная информация о курсе . . . . .	9
1.2 Основные термины в области искусственного интеллекта и машинного обучения . . . . .	11
1.2.1 Общие термины . . . . .	11
1.2.2 Основные специальные термины . . . . .	13
1.2.3 Вопросы для самопроверки . . . . .	16
1.2.4 Резюме по разделу . . . . .	17
1.3 Линейная алгебра: векторы . . . . .	18
1.3.1 Определение вектора . . . . .	18
1.3.2 Операции над векторами . . . . .	19
1.3.3 Реализация в Python . . . . .	28
1.3.4 Вопросы для самопроверки . . . . .	28
1.3.5 Резюме по разделу . . . . .	28
1.4 Линейная алгебра: матрицы . . . . .	29
1.4.1 Определение матрицы . . . . .	29
1.4.2 Операции над матрицами . . . . .	30
1.4.3 Реализация в Python . . . . .	37
1.4.4 Вопросы для самопроверки . . . . .	37
1.4.5 Резюме по разделу . . . . .	38
1.5 Математический анализ: функции . . . . .	40
1.5.1 Определение функции . . . . .	40
1.5.2 Свойства функций . . . . .	40
1.5.3 Графики часто используемых функций . . . . .	41
1.5.4 Реализация в Python . . . . .	47
1.5.5 Вопросы для самопроверки . . . . .	47
1.5.6 Резюме по разделу . . . . .	47
1.6 Математический анализ: производные . . . . .	49
1.6.1 Определение производной . . . . .	49
1.6.2 Недифференцируемые функции . . . . .	51
1.6.3 Правила дифференцирования . . . . .	53
1.6.4 Производные часто используемых функций . . . . .	54
1.6.5 Реализация в Python . . . . .	54
1.6.6 Вопросы для самопроверки . . . . .	55
1.6.7 Резюме по разделу . . . . .	55
1.7 Резюме по модулю . . . . .	56

<b>2 Процесс разработки проекта машинного обучения. Подготовка данных. Оценка качества алгоритмов обучения с учителем. Метод ближайших соседей</b>	<b>59</b>
2.1 Процесс разработки проекта машинного обучения . . . . .	59
2.1.1 Вопросы для самопроверки . . . . .	61
2.1.2 Резюме по разделу . . . . .	61
2.2 Подготовка данных . . . . .	63
2.2.1 Исследование данных . . . . .	63
2.2.2 Очистка и исправление данных . . . . .	63
2.2.3 Разделение выборки на тренировочную и тестовую . . . . .	63
2.2.4 Виды признаков . . . . .	64
2.2.5 Предобработка признаков . . . . .	66
2.2.6 Создание новых признаков . . . . .	72
2.2.7 Отбор признаков . . . . .	72
2.2.8 Реализация в Python . . . . .	73
2.2.9 Вопросы для самопроверки . . . . .	73
2.2.10 Резюме по разделу . . . . .	73
2.3 Оценка качества алгоритмов обучения с учителем . . . . .	75
2.3.1 Метрики качества регрессии . . . . .	75
2.3.2 Метрики качества классификации . . . . .	77
2.3.3 Обобщающая способность алгоритмов и переобучение . . . . .	87
2.3.4 Вопросы для самопроверки . . . . .	88
2.3.5 Резюме по разделу . . . . .	88
2.4 Метод ближайших соседей . . . . .	90
2.4.1 Визуальная демонстрация алгоритма . . . . .	90
2.4.2 Описание алгоритма . . . . .	90
2.4.3 Подготовка данных для алгоритма . . . . .	92
2.4.4 Оценка качества алгоритма . . . . .	92
2.4.5 Модификации алгоритма . . . . .	92
2.4.6 Область применения алгоритма . . . . .	93
2.4.7 Плюсы и минусы алгоритма . . . . .	93
2.4.8 Реализация алгоритма в Python . . . . .	94
2.4.9 Вопросы для самопроверки . . . . .	94
2.4.10 Резюме по разделу . . . . .	95
2.5 Резюме по модулю . . . . .	96
<b>3 Градиентный спуск для обучения дифференцируемых моделей машинного обучения. Методы регуляризации моделей машинного обучения. Линейная регрессия</b>	<b>97</b>
3.1 Градиентный спуск . . . . .	97
3.1.1 Функции потерь и функционалы ошибки в регрессии . . . . .	97
3.1.2 Градиент функции потерь . . . . .	99
3.1.3 Определение градиентного спуска . . . . .	102
3.1.4 Параметры градиентного спуска . . . . .	103
3.1.5 Модификации градиентного спуска . . . . .	105
3.1.6 Вопросы для самопроверки . . . . .	111
3.1.7 Резюме по разделу . . . . .	112

3.2 Регуляризация . . . . .	113
3.2.1 Основные методы регуляризации . . . . .	113
3.2.2 Методы регуляризации, основанные на добавлении слагаемого к функции потерь модели . . . . .	113
3.2.3 Вопросы для самопроверки . . . . .	117
3.2.4 Резюме по разделу . . . . .	117
3.3 Линейная регрессия . . . . .	118
3.3.1 Визуальная демонстрация алгоритма . . . . .	118
3.3.2 Описание алгоритма . . . . .	120
3.3.3 Подготовка данных для алгоритма . . . . .	121
3.3.4 Процесс обучения . . . . .	121
3.3.5 Оценка качества алгоритма . . . . .	124
3.3.6 Интерпретация признаков с помощью алгоритма . . . . .	124
3.3.7 Модификации алгоритма . . . . .	124
3.3.8 Область применения алгоритма . . . . .	125
3.3.9 Плюсы и минусы алгоритма . . . . .	125
3.3.10 Реализация алгоритма в Python . . . . .	126
3.3.11 Вопросы для самопроверки . . . . .	126
3.3.12 Резюме по разделу . . . . .	127
3.4 Резюме по модулю . . . . .	128
<b>4 Линейная классификация. Логистическая регрессия. Метод опорных векторов</b>	<b>129</b>
4.1 Линейная классификация . . . . .	129
4.1.1 Сравнение задач регрессии и классификации . . . . .	129
4.1.2 Трансформация линейной регрессии в линейную классификацию . . . . .	130
4.1.3 Бинарная классификация . . . . .	130
4.1.4 Многоклассовая классификация . . . . .	135
4.1.5 Вопросы для самопроверки . . . . .	138
4.1.6 Резюме по разделу . . . . .	139
4.2 Логистическая регрессия . . . . .	140
4.2.1 Визуальная демонстрация алгоритма . . . . .	140
4.2.2 Описание алгоритма . . . . .	140
4.2.3 Подготовка данных для алгоритма . . . . .	142
4.2.4 Процесс обучения . . . . .	142
4.2.5 Оценка качества алгоритма . . . . .	143
4.2.6 Интерпретация признаков с помощью алгоритма . . . . .	144
4.2.7 Модификации алгоритма . . . . .	144
4.2.8 Область применения алгоритма . . . . .	144
4.2.9 Плюсы и минусы алгоритма . . . . .	145
4.2.10 Реализация алгоритма в Python . . . . .	145
4.2.11 Вопросы для самопроверки . . . . .	146
4.2.12 Резюме по разделу . . . . .	147
4.3 Метод опорных векторов . . . . .	148
4.3.1 Визуальная демонстрация алгоритма . . . . .	148
4.3.2 Описание алгоритма . . . . .	148
4.3.3 Подготовка данных для алгоритма . . . . .	151

4.3.4	Процесс обучения . . . . .	151
4.3.5	Оценка качества алгоритма . . . . .	152
4.3.6	Интерпретация признаков с помощью алгоритма . . . . .	152
4.3.7	Модификации алгоритма . . . . .	153
4.3.8	Область применения алгоритма . . . . .	153
4.3.9	Плюсы и минусы алгоритма . . . . .	154
4.3.10	Реализация алгоритма в Python . . . . .	155
4.3.11	Вопросы для самопроверки . . . . .	155
4.3.12	Резюме по разделу . . . . .	156
4.4	Резюме по модулю . . . . .	156
<b>5</b>	<b>Деревья решений. Композиции деревьев. Случайный лес</b>	<b>159</b>
5.1	Деревья решений . . . . .	159
5.1.1	Описание алгоритма решающего дерева . . . . .	159
5.1.2	Критерии информативности . . . . .	161
5.1.3	Процесс построения дерева для классификации и регрессии . . . . .	163
5.1.4	Регуляризация деревьев решений . . . . .	165
5.1.5	Вопросы для самопроверки . . . . .	167
5.1.6	Резюме по разделу . . . . .	167
5.2	Композиции деревьев . . . . .	168
5.2.1	Подходы к построению композиций . . . . .	168
5.2.2	Бэггинг . . . . .	169
5.2.3	Смещение и разброс . . . . .	170
5.2.4	Вопросы для самопроверки . . . . .	172
5.2.5	Резюме по разделу . . . . .	173
5.3	Случайный лес . . . . .	174
5.3.1	Визуальная демонстрация алгоритма . . . . .	174
5.3.2	Подготовка данных для алгоритма . . . . .	174
5.3.3	Процесс обучения . . . . .	175
5.3.4	Оценка качества алгоритма . . . . .	176
5.3.5	Интерпретация признаков с помощью алгоритма . . . . .	176
5.3.6	Процесс применения . . . . .	176
5.3.7	Область применения алгоритма . . . . .	177
5.3.8	Плюсы и минусы алгоритма . . . . .	177
5.3.9	Модификации алгоритма . . . . .	178
5.3.10	Реализация алгоритма в Python . . . . .	178
5.3.11	Вопросы для самопроверки . . . . .	179
5.3.12	Резюме по разделу . . . . .	180
5.4	Резюме по модулю . . . . .	180
<b>6</b>	<b>Бустинг. Adaboost. Градиентный бустинг. XGBoost. LightGBM. Catboost</b>	<b>181</b>
6.1	Бустинг . . . . .	181
6.1.1	Adaboost . . . . .	182
6.1.2	Градиентный бустинг . . . . .	185
6.1.3	Модификации градиентного бустинга . . . . .	189
6.1.4	Вопросы для самопроверки . . . . .	190
6.1.5	Резюме по разделу . . . . .	191

6.2	XGBoost . . . . .	193
6.2.1	Визуальная демонстрация алгоритма . . . . .	193
6.2.2	Подготовка данных для алгоритма . . . . .	194
6.2.3	Процесс обучения . . . . .	194
6.2.4	Оценка качества алгоритма . . . . .	196
6.2.5	Интерпретация признаков с помощью алгоритма . . . . .	196
6.2.6	Применение алгоритма . . . . .	197
6.2.7	Плюсы и минусы алгоритма . . . . .	197
6.2.8	Реализация алгоритма в Python . . . . .	198
6.2.9	Вопросы для самопроверки . . . . .	198
6.2.10	Резюме по разделу . . . . .	199
6.3	LightGBM . . . . .	200
6.3.1	Визуальная демонстрация алгоритма . . . . .	200
6.3.2	Подготовка данных для алгоритма . . . . .	201
6.3.3	Процесс обучения . . . . .	201
6.3.4	Оценка качества алгоритма . . . . .	203
6.3.5	Интерпретация признаков с помощью алгоритма . . . . .	203
6.3.6	Применение алгоритма . . . . .	203
6.3.7	Плюсы и минусы алгоритма . . . . .	204
6.3.8	Реализация алгоритма в Python . . . . .	204
6.3.9	Вопросы для самопроверки . . . . .	205
6.3.10	Резюме по разделу . . . . .	206
6.4	Catboost . . . . .	207
6.4.1	Визуальная демонстрация алгоритма . . . . .	207
6.4.2	Подготовка данных для алгоритма . . . . .	208
6.4.3	Процесс обучения . . . . .	208
6.4.4	Оценка качества алгоритма . . . . .	209
6.4.5	Интерпретация признаков с помощью алгоритма . . . . .	210
6.4.6	Применение алгоритма . . . . .	210
6.4.7	Плюсы и минусы алгоритма . . . . .	211
6.4.8	Реализация алгоритма в Python . . . . .	211
6.4.9	Вопросы для самопроверки . . . . .	212
6.4.10	Резюме по разделу . . . . .	213
6.5	Резюме по модулю . . . . .	214
7	<b>Обучение без учителя. Кластеризация. Снижение размерности данных</b>	<b>215</b>
7.1	Обучение без учителя . . . . .	215
7.1.1	Кластеризация . . . . .	215
7.1.2	Снижение размерности данных . . . . .	221
7.1.3	Вопросы для самопроверки . . . . .	224
7.1.4	Резюме по разделу . . . . .	224
7.2	K-means . . . . .	226
7.2.1	Визуальная демонстрация алгоритма . . . . .	226
7.2.2	Подготовка данных для алгоритма . . . . .	227
7.2.3	Процесс обучения . . . . .	227
7.2.4	Оценка качества алгоритма . . . . .	228
7.2.5	Применение алгоритма . . . . .	229

7.2.6	Плюсы и минусы алгоритма . . . . .	230
7.2.7	Реализация алгоритма в Python . . . . .	231
7.2.8	Вопросы для самопроверки . . . . .	231
7.2.9	Резюме по разделу . . . . .	232
7.3	DBSCAN . . . . .	233
7.3.1	Визуальная демонстрация алгоритма . . . . .	233
7.3.2	Подготовка данных для алгоритма . . . . .	234
7.3.3	Процесс обучения . . . . .	235
7.3.4	Оценка качества алгоритма . . . . .	236
7.3.5	Применение алгоритма . . . . .	236
7.3.6	Плюсы и минусы алгоритма . . . . .	237
7.3.7	Реализация алгоритма в Python . . . . .	238
7.3.8	Вопросы для самопроверки . . . . .	239
7.3.9	Резюме по разделу . . . . .	239
7.4	Агломеративная кластеризация . . . . .	241
7.4.1	Визуальная демонстрация алгоритма . . . . .	241
7.4.2	Подготовка данных для алгоритма . . . . .	241
7.4.3	Процесс обучения . . . . .	243
7.4.4	Оценка качества алгоритма . . . . .	243
7.4.5	Применение алгоритма . . . . .	244
7.4.6	Плюсы и минусы алгоритма . . . . .	244
7.4.7	Реализация алгоритма в Python . . . . .	245
7.4.8	Вопросы для самопроверки . . . . .	246
7.4.9	Резюме по разделу . . . . .	246
7.5	Метод главных компонент . . . . .	247
7.5.1	Визуальная демонстрация алгоритма . . . . .	250
7.5.2	Подготовка данных для алгоритма . . . . .	254
7.5.3	Процесс обучения . . . . .	254
7.5.4	Оценка качества алгоритма . . . . .	255
7.5.5	Применение алгоритма . . . . .	256
7.5.6	Плюсы и минусы алгоритма . . . . .	257
7.5.7	Реализация алгоритма в Python . . . . .	258
7.5.8	Вопросы для самопроверки . . . . .	258
7.5.9	Резюме по разделу . . . . .	259
7.6	t-SNE . . . . .	260
7.6.1	Визуальная демонстрация алгоритма . . . . .	263
7.6.2	Процесс обучения . . . . .	266
7.6.3	Оценка качества алгоритма . . . . .	267
7.6.4	Применение алгоритма . . . . .	267
7.6.5	Плюсы и минусы алгоритма . . . . .	268
7.6.6	Реализация алгоритма в Python . . . . .	269
7.6.7	Вопросы для самопроверки . . . . .	269
7.6.8	Резюме по разделу . . . . .	270
7.7	Резюме по модулю . . . . .	271



# Глава 1

## Основные термины в машинном обучении. Базовая математика в машинном обучении.

### 1.1 Вводная информация о курсе

Здравствуйте, сегодня мы начинаем изучение курса «Математика и алгоритмы машинного обучения». Замечу, что необходимым условием для успешного прохождения курса является базовое знание языка python и умение пользоваться приложением jupyter notebook для редактирования и запуска кода на python в интерактивном режиме. В этом видео я расскажу вам о том, как устроен наш курс и какие темы мы затронем в ближайшие 7 недель.

Во-первых, каждая неделя будет начинаться с лекционных занятий, на которых будут освещены основные теоретические моменты. Во-вторых, будут практические видео, на которых будут разобраны реализации алгоритмов в различных библиотеках python. Помимо лекций и практических занятий будут тестовые задания. Некоторые из тестовых заданий будут даны для закрепления материала и не будут влиять на итоговую оценку по курсу. Некоторые будут оцениваться и, соответственно, влиять на оценку. Также будут задания на программирование. Эти задания будут выполняться в jupyter notebook. Например, у нас будут задания по работе с матрицами и по обработке обучающих данных, а также на построение тех или иных моделей машинного обучения. Каждое задание на программирование будет сопровождаться подробными инструкциями, что и как нужно делать.

Теперь расскажу об учебном плане курса. На **первой неделе** мы разберемся в основных понятиях в области науки о данных, рассмотрим, какая математика лежит в основе машинного обучения. Мы изучим такие понятия, как векторы, матрицы, функции и производные, которые являются основой для построения алгоритмов машинного обучения.

На последующих неделях я буду рассказывать о признаках, описывающих объекты в наборах данных, о метриках качества для различных алгоритмов, о том, как устроены разные методы машинного обучения, о принципах их работы, различных модификациях и области применения, а также плюсах и минусах.

**Вторую неделю** мы начнем с обсуждения того, какие шаги включает в себя процесс разработки проекта машинного обучения, начиная от определения задачи и сбора данных и заканчивая развертыванием и обслуживанием модели. Кроме того, вторая неделя будет

посвящена различным методам построения признаковых пространств для описания объектов в наборах данных.

Будет рассмотрена тема о метриках качества регрессии в задачах машинного обучения. Также на второй неделе мы начнем изучение класса алгоритмов, называемых обучением с учителем. Обучение с учителем (англ. Supervised learning) - это тип машинного обучения, в котором модель обучается на основе размеченных данных, где каждый пример входных данных сопоставлен с известным, размеченным выходом. Мы поговорим о методе ближайших соседей, одном из самых простых и интуитивно понятных алгоритмов в машинном обучении.

На **третьей неделе** мы разберем такой метод, как линейная регрессия, поговорим о том, что такое переобучение и регуляризация моделей. Кроме этого, мы обсудим такие понятия, как функция потерь и градиентный спуск и как с их помощью численно найти локальный минимум дифференцируемой функции. Градиентный спуск — это основной метод для оптимизации многих алгоритмов в машинном обучении.

**Четвертая неделя** будет посвящена методам линейной классификации. Здесь мы рассмотрим такие модели, как логистическая регрессия и метод опорных векторов. Также на четвертой неделе будут разобраны метрики качества классификации.

На **пятой неделе** мы поговорим о решающих деревьях. Решающее дерево позволяет делать предсказания целевой переменной, используя для этого последовательность простых решающих правил. Такие правила называют предикатами. Шаги по предсказанию ответа с помощью решающего дерева напоминают естественный для человека процесс принятия решений. В лекциях будет рассказано о том, как строить деревья и подбирать предикаты, а также о композициях моделей на основе решающих деревьев.

Темой **шестой недели** будет градиентный бустинг. Градиентный бустинг является одним из наиболее популярных алгоритмов машинного обучения, используемых на практике, наряду с нейронными сетями. Идея бустинга основана на построении композиции из простых моделей (как правило, деревьев) для получения сильной обучающей модели. Алгоритм бустинга заключается в том, что каждая следующая модель в композиции пытается исправить ошибки, допущенные предыдущими моделями, постепенно улучшая качество предсказания. Мы обсудим особенности градиентного бустинга, его настройку и использование.

На **седьмой неделе** мы поговорим о задачах обучения без учителя. Обучение без учителя (англ. Unsupervised learning) - это набор подходов в машинном обучении, при котором модель пытается найти закономерности в неразмеченных данных. Основные задачи, решаемые с помощью обучения без учителя: кластеризация (группировка объектов на основании их похожести), снижение размерности данных (методы PCA, t-SNE) и другие. Мы познакомимся с основными алгоритмами unsupervised learning и их применением на практике.

Мы будем работать с со следующими библиотеками:

- numpy - библиотека python для матричных операций, статистических функций, функций линейной алгебры
- sympy - библиотека python для работы с символьными выражениями, математическими символами и уравнениями
- scipy - библиотека python для работы с массивами, линейной алгеброй, статистикой, оптимизацией
- matplotlib - библиотека python для создания графиков и визуализации данных
- pandas - библиотека python для работы с табличными данными
- scikit-learn - библиотека python для реализации базовых алгоритмов машинного обучения

## 1.2 Основные термины в области искусственного интеллекта и машинного обучения

Прежде чем мы приступим к изучению математики и алгоритмов машинного обучения, давайте рассмотрим основные термины в этой области.

### 1.2.1 Общие термины

- **Линейная алгебра (англ. Linear algebra):** — это раздел математики, который изучает линейные пространства и линейные отображения между ними. Линейная алгебра — основа для многих методов машинного обучения, таких как линейная регрессия, метод главных компонент (PCA), SVM, нейронные сети и многое другое.
- **Математический анализ (англ. Calculus):** — дисциплина, которая занимается изучением функций и их свойств. Математический анализ включает в себя изучение границ, непрерывности, дифференцируемости и интегрирования функций. В машинном обучении используется для оптимизации функций потерь, обучения моделей и других аспектов.
- **Численные методы (англ. Numerical analysis):** это набор методов для приближенного решения математических задач. Они используются в машинном обучении для решения задач оптимизации, аппроксимации функций и т.д.
- **Теория вероятностей и математическая статистика (англ. Probability theory and mathematical statistics):** — дисциплины, которые позволяют оценивать вероятности событий и делать выводы на основе данных. В машинном обучении они используются для создания моделей, оценки их точности, проверки гипотез и т.д.
- **Искусственный интеллект (ИИ, англ. Artificial intelligence, AI)** — это область информатики, которая занимается разработкой алгоритмов и систем, способных выполнять задачи, которые обычно требуют умственных способностей человека, таких как распознавание речи и изображений, обработка естественного языка, принятие решений на основе данных и многие другие. Термин «искусственный интеллект» включает в себя множество областей, таких как машинное обучение, нейронные сети и глубинное обучение. Они используются для создания алгоритмов, которые способны выполнять сложные задачи, которые ранее могли выполнять только люди. Имеются две трактовки искусственного интеллекта: широкий искусственный интеллект и узкий искусственный интеллект. Они отличаются друг от друга по уровню способности к решению различных задач.

**Узкий искусственный интеллект** ограничен в своих способностях и способен решать только определенный набор задач, для которых он был создан. Эти задачи могут включать в себя распознавание речи, обработку естественного языка, игры, автоматизацию задач и т.д. Примеры узкого ИИ включают в себя голосовые помощники Алиса и Siri, которые могут выполнять ограниченный набор задач, таких как открытие приложений, поиск информации в Интернете, установка таймера и т.д.

В идеале, **широкий ИИ** не имеет ограничений и может решать задачу из любой области. На данный момент в мире нет моделей, полностью соответствующих таким характеристикам. Существуют системы, в которых комплексно решается несколько задач узкого искусственного интеллекта. К примеру, это модели, используемые в автономных

автомобилях, роботах-хирургах и системах управления инфраструктурой. Но стоит заметить, что на данный момент появляются большие модели, которые способны решать очень широкий спектр задач, такие как GPT4. В данном курсе будут изучаться базовые модели узкого искусственного интеллекта.

- **Машинное обучение (англ. Machine learning, ML)** — это подраздел искусственного интеллекта, который позволяет компьютерам учиться и делать предсказания на основе анализа больших объемов данных. Машинное обучение использует алгоритмы и статистические модели, чтобы извлекать информацию из данных и делать предсказания или принимать решения. Использование машинного обучения в различных областях, таких как медицина, финансы, промышленность, розничная торговля и другие, позволяет существенно увеличить эффективность и точность принятия решений, а также автоматизировать рутинные задачи.
- **Глубинное обучение (англ. Deep learning)** — это подраздел машинного обучения, который использует нейронные сети с большим количеством слоев для анализа данных и выявления скрытых закономерностей. Основная идея глубинного обучения заключается в том, чтобы создать модель, которая может автоматически извлекать важные признаки из данных, без необходимости ручного определения этих признаков. Для этого глубинные нейронные сети используют многослойную архитектуру, которая позволяет обрабатывать данные на разных уровнях абстракции. В отличие от традиционных методов машинного обучения, где признаки данных должны быть явно определены и предварительно выбраны, глубинное обучение позволяет моделировать данные с большой выразительностью и гибкостью, что делает его особенно полезным для решения сложных задач, таких как распознавание образов, обработка естественного языка, распознавание речи, анализ временных рядов и многих других.
- **Наука о данных (англ. Data science, DS)** — это междисциплинарная область знаний, которая объединяет методы, инструменты и технологии для извлечения знаний и информации из данных. Data science объединяет знания и методы из таких областей, как математика, статистика, информатика, машинное обучение, базы данных и визуализация данных, чтобы анализировать и извлекать знания из больших, разнообразных, сложных и неструктурированных данных.
- **Анализ данных (англ. Data analysis)** - это процесс извлечения полезной информации из данных с целью получения понимания явлений и процессов, лежащих в их основе, а также принятия решений на основе полученных результатов. Анализ данных включает в себя различные методы, такие как статистический анализ, машинное обучение, исследование данных и другие. Целью анализа данных является выявление закономерностей, паттернов и трендов в данных, выделение важных факторов, влияющих на результаты, и прогнозирование будущих событий.
- **Обучение с учителем (англ. Supervised learning)** — тип задач в машинном обучении, при котором модель обучается на размеченных данных, где каждый пример имеет метку класса или значение целевой переменной.
- **Обучение без учителя (англ. Unsupervised learning)** - тип задач в машинном обучении, при котором модель обучается на неразмеченных данных, где нет меток классов или целевой переменной, и цель состоит в том, чтобы выявить структуру данных и паттерны.
- **Обучение с подкреплением (англ. Reinforcement learning, RL)** - тип задач в машинном обучении, при котором модель обучается взаимодействуя с окружающей средой и получая обратную связь в виде награды или штрафа.

### 1.2.2 Основные специальные термины

В машинном обучении также существует множество специальных терминов, которые описывают различные концепции, методы и алгоритмы. В первую очередь, приведем основные из них.

#### Основные компоненты, используемые в алгоритмах машинного обучения

- **Объекты (англ. Objects)** — наблюдаемые данные, которые необходимо обработать и проанализировать. Объекты могут представлять собой любые данные, которые можно описать числами или другими количественными характеристиками, такими как звуковые сигналы, изображения, текстовые документы, числовые таблицы и т.д. Каждый объект в машинном обучении представлен набором признаков или характеристик, которые описывают его свойства и состояние. Например, для изображения объектом могут быть пиксели, а для текстового документа - слова или символы. Объекты в машинном обучении используются для обучения моделей, которые могут классифицировать или регрессировать новые объекты на основе обучающих данных. Кроме того, объекты могут использоваться для оценки качества моделей и проведения исследовательского анализа данных.
- **Признаки (англ. Features)** — измерения или характеристики объекта, которые используются для его описания и анализа. Признаки могут быть числовыми, бинарными или категориальными, в зависимости от типа данных, которые они описывают. Например, если объектом является человек, то признаками могут быть его возраст, рост, вес, пол, образование, занятость и т.д. Для изображения признаками могут быть яркость и цвет каждого пикселя, а для текста - количество вхождений каждого слова. Признаки играют важную роль в машинном обучении, поскольку на их основе строятся модели для решения задач классификации, регрессии, кластеризации и т.д. Выбор и определение правильных признаков является важным шагом в обработке и подготовке данных, поскольку неинформативные или некорректно выбранные признаки могут привести к плохим результатам моделирования.
- **Целевая переменная (англ. Target variable)** — это переменная, которую модель пытается предсказать или прогнозировать на основе других переменных, называемых признаками или характеристиками. Целевая переменная может быть числовой или категориальной, в зависимости от типа задачи, которую решает модель. В задачах регрессии, целевая переменная представляет собой непрерывную числовую переменную, которую модель пытается предсказать. В задачах классификации, целевая переменная представляет собой категориальную переменную, которую модель пытается определить для новых данных. Выбор правильной целевой переменной является критически важным в машинном обучении, так как это определяет тип задачи и алгоритмы, которые будут использоваться для обучения модели.
- **Набор данных (англ. Dataset)** — это коллекция данных, которые используются для обучения и тестирования моделей машинного обучения. Набор данных может содержать множество примеров, каждый из которых представляет собой некоторый объект или явление, для которых уже известны значения признаков и целевых переменных, которые нужно предсказать. Для построения модели машинного обучения требуется набор данных, который будет использоваться для обучения (training dataset) и для проверки качества модели

(validation dataset или test dataset). Обычно набор данных разбивается на две части: одна часть используется для обучения модели, а другая - для ее проверки. Размеры частей могут варьироваться, но обычно на обучение выделяется от 70% до 90% данных, а на проверку - от 10% до 30%.

Набор данных должен быть предварительно обработан и очищен от выбросов, пропусков и ошибок. Также может потребоваться преобразование данных для улучшения их качества или для адаптации под конкретный алгоритм машинного обучения. Наборы данных могут быть различных типов: текстовые, изображений, звуковые, видео и другие. Они могут быть собраны из различных источников: измерений, экспериментов, интернета и т.д.

- **Модель** — это математическое представление, которое используется для решения конкретной задачи, например, для предсказания значения целевой переменной на основе данных признаков. Модель обучается на наборе данных, который включает в себя объекты, признаки и (в задачах регрессии и классификации) целевую переменную, и после этого может быть использована для предсказания для новых данных.

Модель представлена в виде алгоритма, который может быть реализован на компьютере. Обучение модели в машинном обучении заключается в том, чтобы настроить ее параметры таким образом, чтобы минимизировать ошибку прогноза на данных. Этот процесс обучения может быть выполнен различными способами, в зависимости от типа задачи и выбранного алгоритма.

Существует множество различных типов моделей, используемых в машинном обучении, включая метрические модели, линейные модели, деревья решений, ансамблевые модели, нейронные сети и другие. Каждая модель имеет свои сильные и слабые стороны, и выбор конкретной модели зависит от требуемой точности, сложности задачи, объема данных и других факторов.

- **Функция потерь (англ. Loss function)** — математическая функция, которая оценивает разницу между предсказанными значениями модели и фактическими значениями на обучающем наборе данных. Цель функции потерь заключается в том, чтобы минимизировать разницу между предсказанными значениями и фактическими значениями, чтобы модель могла давать наиболее точные прогнозы на новых данных. В задачах регрессии, когда требуется предсказать численное значение (например, цену недвижимости), функция потерь может быть выбрана как среднеквадратическая ошибка (MSE) или средняя абсолютная ошибка (MAE). В задачах классификации, когда требуется предсказать категорию или метку, функция потерь может быть выбрана как перекрестная энтропия (cross-entropy) или логистическая функция потерь.

В машинном обучении выбор функции потерь зависит от задачи, типа модели и используемого алгоритма обучения. Выбор правильной функции потерь может повлиять на точность модели и скорость ее обучения. Однако, в некоторых случаях может быть необходимо оптимизировать более сложную функцию потерь, которая учитывает не только разницу между предсказанными и фактическими значениями, но и другие факторы, такие как сложность модели или балансировку классов в задаче классификации.

**Математическая нотация** приведенных выше определений следующая:

- $x_i$  - объект, для которого делается предсказание
- $d$  - размерность
- $\mathbb{X}$  - пространство объектов
- $(x_{i1}, x_{i2}, \dots, x_{id})$  - набор из  $d$  признаков для  $i$ -го объекта

- $y_i$  - ответ для объекта  $i$ , целевая переменная
- $\mathbb{Y}$  - пространство ответов
- $n$  - количество примеров, размер обучающей выборки
- $X = (x_i, y_i)_{i=1}^n$  - обучающая выборка
- $m(x_i)$  - модель - функция, предсказывающая ответ для объекта  $x_i$ . Можно сказать, что  $m$  отображает пространство  $\mathbb{X}$  в пространство  $\mathbb{Y}$
- $\hat{y}_i$  - предсказание модели для объекта  $i$
- $L(y_i, \hat{y}_i)$  - функция потерь - показатель того, насколько предсказание модели отличается от реального ответа на конкретном примере

## Основные типы задач в машинном обучении

Давайте разберемся в основных типах задач в машинном обучении.

- **Регрессия (англ. Regression)** — это метод, используемый для анализа отношений между целевой переменной (иногда её называют зависимой переменной) и одной или несколькими независимыми переменными (признаками). Она часто используется для прогнозирования количественных значений, таких как цены, объемы продаж, количество посетителей и т.д.

В задаче регрессии целевая переменная является количественной, то есть она может принимать любые числовые значения. Цель регрессионного анализа - построить модель, которая может предсказывать значения целевой переменной на основе значений признаков. Для построения модели регрессии используется алгоритм, который находит оптимальные значения параметров модели, минимизирующие ошибку прогнозирования на обучающих данных. Существует множество алгоритмов регрессии, включая метрические методы, линейную регрессию, деревья решений и другие.

Одной из важных задач в регрессии является оценка качества модели на новых данных, для чего используется тестовый набор данных. Для оценки качества модели применяются различные метрики, такие как среднеквадратичная ошибка (MSE), средняя абсолютная ошибка (MAE) и другие.

- **Классификация (англ. Classification)** — это процесс отнесения объектов к заранее определенным классам или категориям. Классификация используется в широком спектре задач, таких как распознавание образов, определение темы текста, диагностика заболеваний, прогнозирование рынка и многих других.

В задачах классификации используются алгоритмы машинного обучения, которые обучаются на наборе данных, состоящем из объектов и соответствующих им меток классов (целевых переменных). Обычно для обучения модели используется алгоритм, который находит зависимость между признаками объектов и их метками классов, а затем использует эту зависимость для прогнозирования меток классов для новых объектов.

Примерами алгоритмов классификации являются логистическая регрессия, метод опорных векторов (SVM), деревья решений, градиентный бустинг и нейронные сети. В зависимости от природы задачи и характеристик данных, один алгоритм может быть более эффективным, чем другой.

Для оценки качества работы алгоритма классификации используются различные метрики, такие как доля верно предсказанных ответов (accuracy), полнота (recall), точность предсказания (precision), F1-мера и другие. Для выбора наилучшего алгоритма классификации используется оценка метрики на тестовом датасете или кросс-валидация на обучающих данных.

- **Кластеризация** (англ. *Clusterization*) — это задача разбиения множества объектов на группы (кластеры) таким образом, чтобы объекты внутри каждого кластера были максимально похожи между собой, а объекты из разных кластеров были максимально различными.

Кластеризация используется во многих областях, таких как маркетинг, медицина, социология, биология и другие. Для выполнения задачи кластеризации используются различные алгоритмы, такие как K-средних (K-means), DBSCAN, иерархическая кластеризация и т.д. Алгоритмы кластеризации могут работать с различными типами данных, включая числовые, категориальные и текстовые.

Оценка качества кластеризации может быть достаточно сложной задачей, так как нет точного определения того, как должны быть сформированы кластеры. Для оценки качества кластеризации используются различные метрики, такие как коэффициент силуэта, индекс Данна и другие.

- **Снижение размерности** (англ. *Dimensionality reduction*) — это процесс уменьшения количества признаков в наборе данных, при этом сохраняется максимально возможное количество информации о данных. Это позволяет упростить анализ данных и ускорить вычисления в алгоритмах машинного обучения. Снижение размерности может производиться как для уменьшения количества шума и выбросов в данных, так и для упрощения модели при сохранении информативных признаков. Снижение размерности может быть полезным во многих задачах машинного обучения, таких как классификация, кластеризация, визуализация данных и других.

Помимо приведенных выше типов задач в машинном обучении встречаются и другие (обучение с подкреплением, детектирование аномалий и пр.). Их изучение выходит за рамки тем данного курса.

### 1.2.3 Вопросы для самопроверки

Математический анализ - это ...:

1. раздел математики, который изучает линейные пространства и линейные отображения между ними
2. раздел математики, который изучает функции и их свойства
3. набор методов численного решения математических задач
4. раздел математики, который изучает вероятности событий

Правильные ответы: 2

Как соотносятся понятия "глубинное обучение" "искусственный интеллект" и "машинное обучение" (от большего множества к меньшему):

1. глубинное обучение -> искусственный интеллект -> машинное обучение
2. машинное обучение -> глубинное обучение -> искусственный интеллект
3. искусственный интеллект -> глубинное обучение -> машинное обучение
4. искусственный интеллект -> машинное обучение -> глубинное обучение

Правильные ответы: 4

Для какого типа задач в машинном обучении необходимы метки с правильными ответами для обучения:

1. Обучение с учителем (англ. Supervised learning)
2. Обучение без учителя (англ. Unsupervised learning)
3. Обучение с подкреплением (англ. Reinforcement learning, RL)

Правильные ответы: 1

#### 1.2.4 Резюме по разделу

В этом разделе мы поговорили об основных разделах математики, которые используются в машинном обучении. Мы познакомились с перечнем наиболее часто применяемых в машинном обучении разделов математики: линейная алгебра, математический анализ, оптимизация, численные методы, теория вероятностей, теория графов, теория информации. Математический аппарат позволяет создавать новые методы машинного обучения, представлять методы машинного обучения в виде математических моделей, выражать данные в нужном виде и оптимизировать различные функции.

Мы также рассмотрели основные общие термины в машинном обучении, а именно искусственный интеллект, машинное обучение, глубинное обучение, наука о данных, анализ данных, обучение с учителем, обучение без учителя, обучение с подкреплением. Также мы познакомились с основными специальными терминами, такими как объекты, признаки, целевая переменная, набор данных, модель, функция потерь, регрессия, классификация, кластеризация, снижение размерности.

## 1.3 Линейная алгебра: векторы

### 1.3.1 Определение вектора

Вектор — это математический объект, который имеет длину и направление в пространстве. Он может быть использован для представления физических величин, таких как сила, скорость, ускорение и многих других. В машинном обучении векторы являются одним из основных понятий и используются для представления данных. Например, вектор может представлять цвет изображения в RGB-формате, размер объекта на изображении, частоты звука или другие характеристики данных. Векторы также используются в алгоритмах машинного обучения для выполнения операций, таких как классификация, кластеризация и регрессия. Кроме того, векторы могут использоваться для решения задач оптимизации. Каждый объект может быть представлен в виде вектора признаков, где каждый элемент вектора соответствует определенному признаку объекта.

Векторы могут быть обработаны с помощью различных методов машинного обучения, таких как линейная регрессия, метод опорных векторов, нейронные сети, деревья решений и многие другие. При обучении моделей машинного обучения векторы используются для поиска закономерностей, связей и корреляций между признаками, которые могут быть использованы для прогнозирования или классификации новых данных. Векторы используются для создания матрицы признаков, которая является основным входом для большинства алгоритмов машинного обучения. В python распространенный способ представления вектора - это одномерный массив numpy.

Векторы обычно обозначаются либо с помощью жирного шрифта, либо с помощью стрелки над символом:  $\mathbf{v}$  или  $\vec{v}$ . В общем случае, в  $n$ -мерном пространстве вектор представляется как набор из  $n$  чисел. Компоненты вектора могут быть действительными или комплексными числами. Для записи компонент вектора обычно используют формат столбца (вектор-столбец), где каждый элемент записывается в новой строке:

$$\mathbf{v} = \vec{v} = \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix}$$

Элементы вектора  $v_1, v_2, v_3, \dots, v_{n-1}, v_n$  называются его **координатами** или **компонентами**. Размерностью вектора является количество его компонент.

Существует связь между понятием вектора в  $n$ -мерном пространстве и упорядоченной последовательностью действительных чисел  $v_1, v_2, v_3, \dots, v_{n-1}, v_n$ . В двумерном пространстве, к примеру, вектор может быть представлен как пара чисел  $(v_1, v_2)$ , где  $v_1, v_2$  могут быть интерпретированы как координаты  $x, y$  на плоскости (Рисунок 1.1 (1)). В трехмерном пространстве векторы задаются с помощью трех координат (Рисунок 1.1 (2)). Векторы большей размерности не могут быть визуализированы.

Некоторые типы векторов имеют собственные названия. Например, нулевой вектор — это вектор, все компоненты которого равны нулю:

$$\vec{0} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix}$$

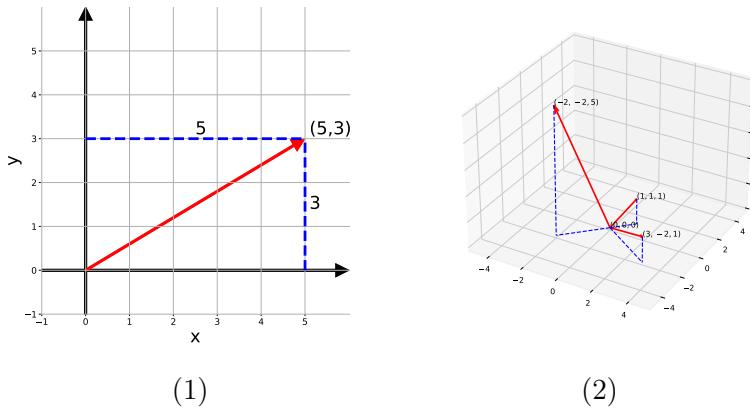


Рисунок 1.1: (1) — Вектор с координатами  $(5,3)$  в двумерном пространстве, (2) — несколько векторов в трехмерном пространстве

В **машинном обучении** принято задавать векторы с помощью только одной точки в заданном векторном пространстве. Соответственно, полагается, что **начало всех векторов находится в точке начала координат**. Такая интерпретация векторов делает интуитивно понятными различные операции над векторами, например, вычисление угла между двумя векторами (Рисунок 1.2).

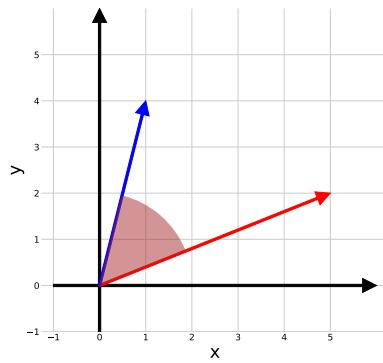


Рисунок 1.2: Два вектора, исходящих из начала координат

Рассмотрим основные операции над векторами и их геометрический смысл.

### 1.3.2 Операции над векторами

## Определение направления вектора

Направление вектора определяется по направлению от его начала к концу. Другими словами, направление вектора - это угол, под которым он направлен относительно некоторой начальной точки (обычно начало координат) в пространстве. Обычно угол измеряется в градусах или радианах. Например, для вектора с координатами  $(5,4)$ , угол между горизонтальной осью и вектором может быть найден, используя тангенс угла наклона, который равен  $4/5$ . Чтобы определить угол по тангенсу, можно воспользоваться обратной функцией тангенса -

арктангенсом (или тангенсом $^{-1}$ ). Обозначается арктангенс как  $\arctan$  или  $\tan^{-1}$ . Если дано значение тангенса угла  $\alpha$ , то угол  $\alpha$  можно найти следующим образом:

$$\alpha = \arctan(\tan \alpha)$$

Например, если дано, что  $\tan \alpha = 4/5$ , то угол  $\alpha$  можно найти следующим образом:

$$\alpha = \arctan(0.8) \approx 38.659^\circ$$

Однако необходимо помнить, что арктангенс имеет ограничения на область определения и может возвращать только углы в определенном диапазоне значений. Например, для обычной арктангенс функции область определения ограничена значениями  $-\frac{\pi}{2} < \arctan(x) < \frac{\pi}{2}$ , то есть функция возвращает только углы, лежащие в этом диапазоне. Если необходимо получить угол, лежащий в другом диапазоне, можно использовать дополнительные математические операции, например, добавлять или вычитать  $\pi$  (180 градусов) из значения арктангенса в зависимости от того, в какой четверти находится искомый угол.

Соответственно, направление вектора с координатами  $(5,4)$  равно  $\approx 38.659^\circ$  (Рисунок 1.3).

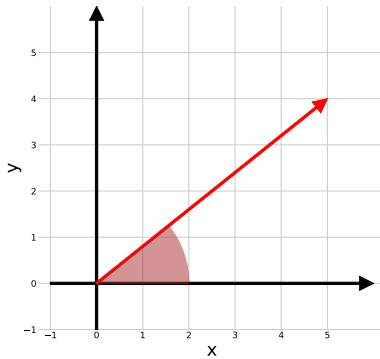


Рисунок 1.3: Направление вектора, выраженное в градусах

## Вычисление длины вектора

Каждый вектор в векторном пространстве, кроме 0-вектора, имеет длину, длина обозначается термином *норма*. Существуют несколько способов определения нормы векторов, но все они моделируют расстояния, которые мы используем в повседневной жизни. Один из способов - это **L1-норма** (Расстояние городских кварталов, Манхэттенская метрика), которая соответствует расстоянию, которое бы проехал автомобиль между начальной и конечной точками по городским улицам, расположенным под углом  $90^\circ$ . Это расстояние находится путем перемещения только вдоль осей координат (Формула 1.1):

$$\|\vec{v}\|_1 = \left( \sum_{i=1}^d |v_i|^1 \right)^{\frac{1}{1}}, \quad (1.1)$$

где  $v$  - вектор размерности  $d$

В этой формуле происходит суммирование модулей всех координат. Корень 1-й степени означает отсутствие математических действий над выражением и используется лишь для унификации с метрикой Минковского (Формула 1.3). К примеру, расстояние от точки  $(0,0)$  до

точки  $(5,4)$  может быть пройдено по разным траекториям, но общее расстояние останется равным 9.

Другой способ - это **L2-норма**, которая соответствует расстоянию, которое пролетел бы вертолет между началом координат и точкой вектора. Это расстояние находится по теореме Пифагора. Сначала происходит суммирование квадратов всех координат, а затем вычисляется корень 2-й степени из этой суммы (Формула 1.2):

$$\|\vec{v}\|_2 = \left( \sum_{i=1}^d |v_i|^2 \right)^{\frac{1}{2}}, \quad (1.2)$$

где  $v$  - вектор размерности  $d$

L2-норма является более популярной по сравнению с L1-нормой, потому что она более естественна для измерения длины (Рисунок 1.4).

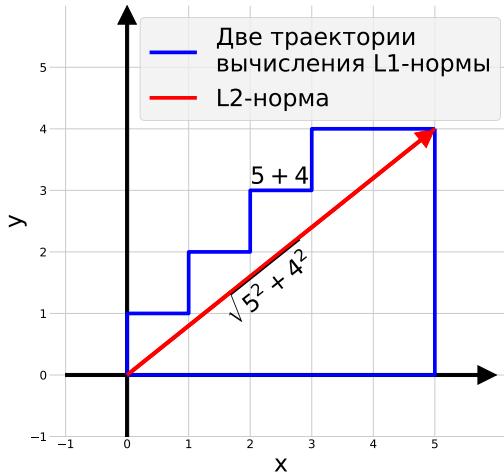


Рисунок 1.4: Способы вычисления длины вектора

Общая же формула нормы вектора в  $\mathbb{R}^d$  носит название метрики Минковского и вычисляется по формуле (Формула 1.3):

$$\|\vec{v}\|_n = \left( \sum_{i=1}^d |v_i|^n \right)^{\frac{1}{n}}, \text{ где } v \text{ - вектор размерности } d, n \text{ - показатель степени} \quad (1.3)$$

Поясним, что запись  $\mathbb{R}^d$  означает, что мы рассматриваем векторы, элементы которых являются числами из множества действительных чисел  $\mathbb{R}$ , и каждый вектор имеет  $d$  компонент (или размерность  $d$ ). Таким образом, векторы в  $\mathbb{R}^d$  можно записать в виде упорядоченных наборов чисел вида  $(x_1, x_2, \dots, x_d)$ , где каждое  $x_i$  принадлежит множеству действительных чисел  $\mathbb{R}$ . Примеры:

- Вектор  $(2, 5, -3)$  принадлежит векторному пространству  $\mathbb{R}^3$ .
- Вектор  $(1, 0, 1, -2)$  принадлежит векторному пространству  $\mathbb{R}^4$ .

## Сложение и вычитание векторов

Векторы, так же как и числа, можно складывать:

$$\vec{v1} + \vec{v2} = \begin{bmatrix} v1_1 \\ v1_2 \\ \vdots \\ v1_n \end{bmatrix} + \begin{bmatrix} v2_1 \\ v2_2 \\ \vdots \\ v2_n \end{bmatrix} = \begin{bmatrix} v1_1 + v2_1 \\ v1_2 + v2_2 \\ \vdots \\ v1_n + v2_n \end{bmatrix}$$

и вычитать друг из друга, получая новые векторы:

$$\vec{v1} + (-\vec{v2}) = \begin{bmatrix} v1_1 \\ v1_2 \\ \vdots \\ v1_n \end{bmatrix} + \begin{bmatrix} -v2_1 \\ -v2_2 \\ \vdots \\ -v2_n \end{bmatrix} = \begin{bmatrix} v1_1 - v2_1 \\ v1_2 - v2_2 \\ \vdots \\ v1_n - v2_n \end{bmatrix}$$

Например, для того чтобы сложить вектор  $\vec{v1}$  с координатами (4,7) (на рисунке 1.5 (1) красного цвета) и вектор  $\vec{v2}$  с координатами (8,4) (на рисунке 1.5 (1) синего цвета), нужно просто сложить их соответствующие координаты, и тогда получится новый вектор  $v1 + v2$  с координатами (12,11) (на рисунке 1.5 (1) зеленого цвета). Геометрически это можно представить так: суммарный вектор является диагональю параллелограмма, образованного векторами  $\vec{v1}$  и  $\vec{v2}$  (Рисунок 1.5 (1)).

Вычитание происходит аналогичным образом, только используется другая диагональ параллелограмма. Например, разность между векторами  $\vec{v1}$  (4,7) (на рисунке 1.5 (2) красного цвета) и  $\vec{v2}$  (8,4) (на рисунке 1.5 (2) синего цвета) вычисляется также поэлементно, получается вектор  $v1 - v2$  (на рисунке 1.5 (2) зеленого цвета) с координатами (-4,3) (Рисунок 1.5 (2)):

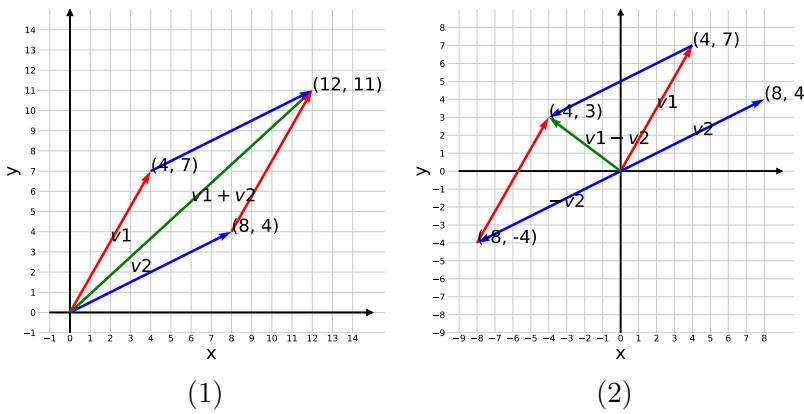


Рисунок 1.5: (1) — Сложение векторов, (2) — вычитание векторов

### Свойства сложения векторов:

Для любых векторов одинакового размера выполнено:

- $\vec{v1} + \vec{v2} = \vec{v2} + \vec{v1}$  (коммутативность сложения)
- $(\vec{v1} + \vec{v2}) + \vec{v3} = \vec{v1} + (\vec{v2} + \vec{v3})$  (ассоциативность сложения)

### Умножение вектора на константу

Умножение вектора на константу — это поэлементное умножение компонент вектора на данную константу.

$$c\vec{v} = c \begin{bmatrix} v_1 \\ v_2 \\ \vdots \\ v_n \end{bmatrix} = \begin{bmatrix} cv_1 \\ cv_2 \\ \vdots \\ cv_n \end{bmatrix}$$

Умножение вектора на константу приводит к изменению длины вектора и/или его направления, в зависимости от значения константы. Геометрический смысл умножения вектора на константу следующий (Рисунок 1.6):

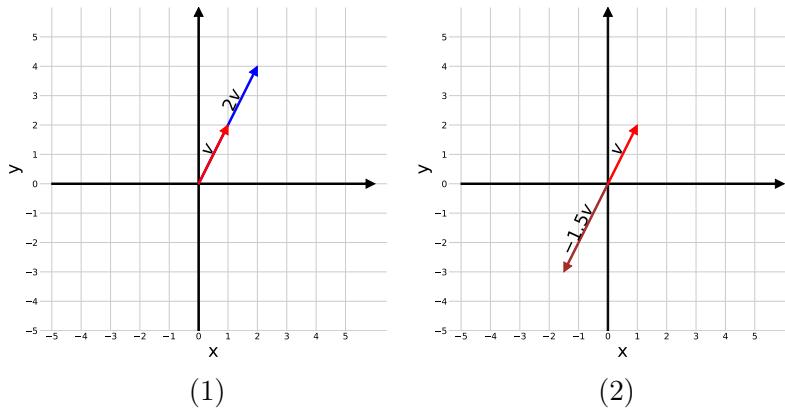


Рисунок 1.6: (1) — константа положительная, умножение приводит к увеличению длины вектора, направление не меняется, (2) — константа отрицательная, длина вектора также увеличивается, но направление меняется на противоположное

Умножение вектора на число может использоваться для масштабирования векторов в графическом программировании, изменения скорости или ускорения объектов в физических симуляциях, применения линейных преобразований в линейной алгебре, и т.д.

#### Свойства умножения вектора на константу:

Для любого вектора выполнено:

- $c \cdot \vec{v} = \vec{v} \cdot c$  (коммутативность умножения на константу)
- $(c_1 c_2) \cdot \vec{v} = c_1 \cdot (c_2 \cdot \vec{v})$  (ассоциативность умножения на несколько констант)

### Транспонирование вектора

Для вектора применима операция транспонирования. Транспонирование вектора - это операция, при которой вектор записывается в виде матрицы размера  $1 \times n$  (если вектор-столбец) или  $n \times 1$  (если вектор-строка), где  $n$  - количество элементов вектора.

При транспонировании вектор-столбца получается вектор-строка:

$$\vec{v} = [v_1 \quad v_2 \quad \cdots \quad v_n]$$

## Скалярное произведение векторов

Скалярное произведение векторов - это операция, результатом которой является скалярное значение (число), полученное путем умножения соответствующих элементов двух векторов и их последующей суммы.

Скалярное произведение может быть вычислено, только если длины двух векторов равны.

Для скалярного произведения векторов  $\vec{v}_1$  и  $\vec{v}_2$  используется одно из следующих обозначений:  $\langle \vec{v}_1, \vec{v}_2 \rangle$ ,  $(\vec{v}_1, \vec{v}_2)$ ,  $\mathbf{v}_1 \cdot \mathbf{v}_2$ ,  $\vec{v}_1 \cdot \vec{v}_2$ .

Для двух векторов  $\vec{v}_1$  и  $\vec{v}_2$  длиной  $n$  скалярное произведение может быть вычислено по формуле 1.4:

$$\langle \vec{v}_1, \vec{v}_2 \rangle = \sum_{i=1}^n v_{1i} v_{2i}, \quad (1.4)$$

где  $\vec{v}_1 = [v_{11}, v_{12}, \dots, v_{1n}]$  и  $\vec{v}_2 = [v_{21}, v_{22}, \dots, v_{2n}]$  - два вектора длины  $n$ .

Скалярное произведение также может быть записано в виде произведения вектора-строки на вектор-столбец.

### Свойства скалярного произведения векторов:

- С помощью скалярного произведения может быть вычислена длина вектора  $\vec{v}$  как корень из скалярного произведения вектора на себя:  $L_2$ -norm =  $\sqrt{\langle \vec{v}, \vec{v} \rangle}$ .
- С помощью скалярного произведения может быть вычислен угол между двумя векторами  $\vec{v}_1$  и  $\vec{v}_2$ :  $\cos \theta = \frac{\langle \vec{v}_1, \vec{v}_2 \rangle}{|\vec{v}_1| |\vec{v}_2|}$ , где  $\theta$  - угол между векторами,  $|\vec{v}_1|$  и  $|\vec{v}_2|$  - длины векторов  $\vec{v}_1$  и  $\vec{v}_2$  соответственно.
- С помощью скалярного произведения может быть вычислена проекция вектора  $\vec{v}_2$  на вектор  $\vec{v}_1$ :  $\text{proj}_{\vec{v}_1} \vec{v}_2 = \frac{\langle \vec{v}_1, \vec{v}_2 \rangle}{|\vec{v}_1|^2} \vec{v}_1$  (Рисунок 1.7):

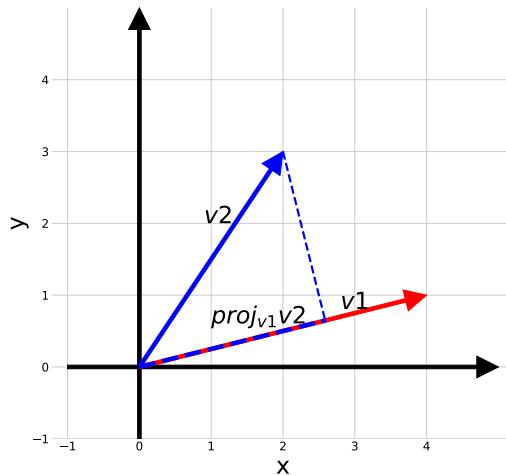


Рисунок 1.7: Проекция вектора  $v_2$  на вектор  $v_1$

- Для всех векторов, угол между которыми меньше  $90^\circ$ , скалярное произведение будет положительным, для ортогональных векторов (угол между ними равен  $90^\circ$ ) скалярное произведение равно 0, для всех векторов, угол между которыми больше  $90^\circ$ , скалярное произведение будет отрицательным (Рисунок 1.8).

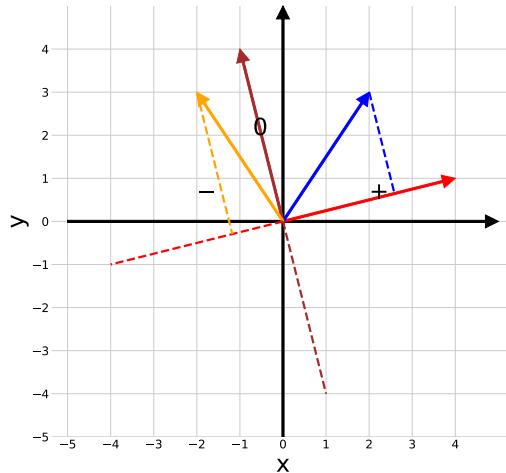


Рисунок 1.8: Знак скалярного произведения при разных углах между векторами

Скалярное произведение векторов имеет множество применений в математике, физике и инженерии. Некоторые из них:

- Решение систем линейных уравнений: система линейных уравнений может быть записана в виде матричного уравнения  $\mathbf{Ax} = \mathbf{b}$ , где  $\mathbf{A}$  - матрица коэффициентов,  $\mathbf{x}$  - вектор неизвестных,  $\mathbf{b}$  - вектор правых частей. Решение этой системы может быть получено с помощью методов, основанных на скалярном произведении, например, метод Гаусса.
- Кластерный анализ: в кластерном анализе скалярное произведение может быть использовано для измерения сходства между объектами или для оценки близости кластеров.

### Векторное произведение векторов

Векторное произведение - это бинарная операция над двумя векторами, результатом которой является новый вектор, перпендикулярный обоим исходным векторам. На рисунке 1.9 показано векторное произведение вектора  $\vec{v}_1$  с координатами  $(2, -2, 0)$  (красный цвет) на вектор  $\vec{v}_2$  с координатами  $(1, 2, 0)$  (синий цвет). Результатом является вектор  $v_1 \times v_2$  с координатами  $(0, 0, 6)$  (зеленый цвет).

Компоненты вектора, полученного в результате векторного произведения, вычисляются как разности попарных произведений компонент исходных векторов, умноженных на коэффициенты  $\pm 1$  в соответствии с правилом буравчика<sup>1</sup>. Векторное произведение двух векторов  $\vec{v}_1$  и  $\vec{v}_2$  в трехмерном пространстве вычисляется по формуле 1.5:

$$\vec{a} \times \vec{b} = (a_1 a_2 a_3) \times (b_1 b_2 b_3) = (a_2 b_3 - a_3 b_2 a_3 b_1 - a_1 b_3 a_1 b_2 - a_2 b_1), \quad (1.5)$$

где  $\vec{a} = (a_1, a_2, a_3)$  и  $\vec{b} = (b_1, b_2, b_3)$  - два вектора в  $\mathbb{R}^3$ .

<sup>1</sup>Правило буравчика (винта) для векторного произведения: «Если отобразить векторы так, чтобы их начала совпадали и вращать первый вектор-сомножитель кратчайшим образом ко второму вектору-сомножителю, то буравчик (винт), вращающийся таким же образом, будет завинчиваться в направлении вектора-произведения».

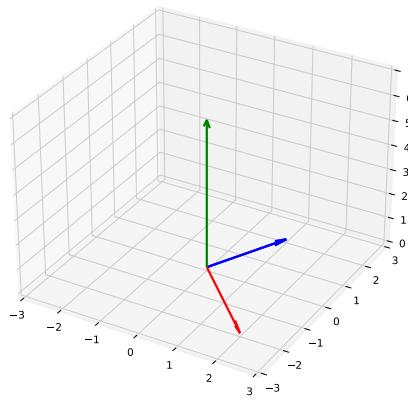


Рисунок 1.9: Векторное произведение векторов

Векторное произведение используется в машинном обучении довольно редко, но все же может быть полезным для некоторых задач, например, для создания новых признаков из двух или более существующих признаков.

### Вычисление расстояния между векторами

Определение расстояния между двумя векторами имеет важное применение в машинном обучении. Для функции измерения расстояния между двумя векторами используется термин метрика. Если мы используем метрику для измерения расстояния между векторами, то она должна удовлетворять следующим требованиям:

- Неотрицательность: Метрика должна быть неотрицательной, то есть расстояние между любыми двумя векторами должно быть не меньше нуля. Формально:  $d(x,y) \geq 0$  для любых векторов  $x$  и  $y$ , и  $d(x,y) = 0$  тогда и только тогда, когда  $x = y$ .
- Симметричность: Метрика должна быть симметричной, то есть расстояние между векторами  $x$  и  $y$  должно быть таким же, как расстояние между  $y$  и  $x$ . Формально:  $d(x,y) = d(y,x)$  для любых векторов  $x$  и  $y$ .
- Неравенство треугольника: Метрика должна удовлетворять неравенству треугольника, то есть расстояние между любыми двумя векторами, через третий вектор, должно быть меньше, чем сумма расстояний между первым и третьим векторами, и между вторым и третьим векторами. Формально:  $d(x,y) \leq d(x,z) + d(z,y)$  для любых векторов  $x$ ,  $y$  и  $z$ .
- Тождественность: Метрика должна удовлетворять тождеству:  $d(x,y) = 0$  тогда и только тогда, когда  $x = y$ .

Метрика, удовлетворяющая этим требованиям, называется метрикой расстояния.

Мы можем использовать способы вычисления расстояния, уже рассмотренные для определения длины вектора. В случае двух векторов L1 расстояние вычисляется как расстояние

между концами векторов. Если говорить строго, то такое расстояние называется суммой абсолютных значений разности компонент. Оно вычисляется по формуле 1.6:

$$|\vec{v1} - \vec{v2}|_1 = \sum_{i=1}^n |v1_i - v2_i|, \quad (1.6)$$

где  $\vec{v1}$  и  $\vec{v2}$  - два вектора размерности  $n$ ,

$v1_i$  и  $v2_i$  - их соответствующие компоненты.

Другой способ - L2-метрика или L2-расстояние - вычисляется как корень квадратный из суммы квадратов разностей компонент по формуле 1.7:

$$|\vec{v1} - \vec{v2}|_2 = \sqrt{\sum_{i=1}^n (v1_i - v2_i)^2}, \quad (1.7)$$

где  $\vec{v1}$  и  $\vec{v2}$  - два вектора размерности  $n$ ,

$v1_i$  и  $v2_i$  - их соответствующие компоненты.

Косинусная метрика или косинусное расстояние - это еще один способ определения различий между векторами, вычисляется как 1 - косинус угла между ними по формуле 1.8:

$$\text{cosine distance}(\vec{v1}, \vec{v2}) = 1 - (\vec{v1} \cdot \vec{v2}) / (|\vec{v1}| * |\vec{v2}|),$$

где  $\vec{v1} \cdot \vec{v2}$  - скалярное произведение векторов  $\vec{v1}$  и  $\vec{v2}$ ,  
 $|\vec{v1}|$  и  $|\vec{v2}|$  - длины векторов  $\vec{v1}$  и  $\vec{v2}$

В машинном обучении знание этих различных мер расстояния между векторами очень полезно для определения сходства между точками данных. Визуальное представления различных мер расстояния показано на рисунке 1.10:

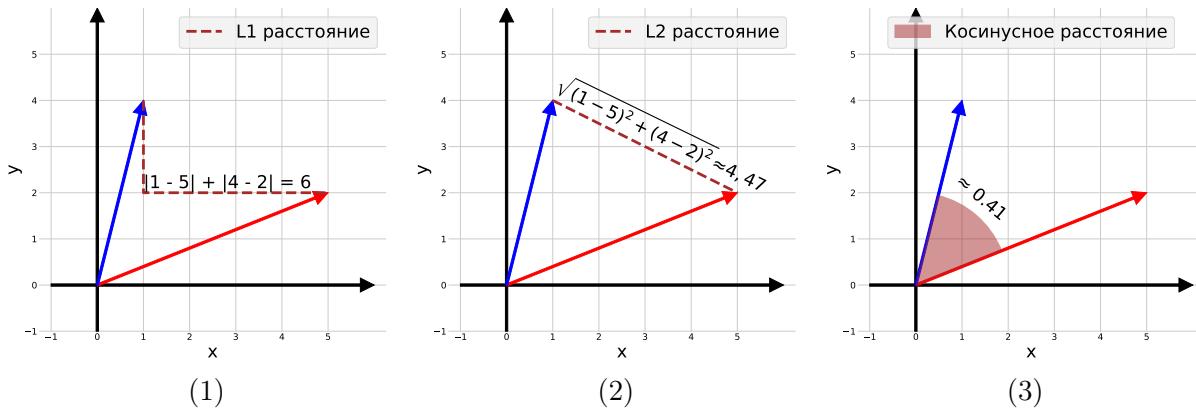


Рисунок 1.10: (1) — L1-расстояние, (2) — L2-расстояние, (3) — косинусное расстояние

L1-расстояние может быть предпочтительнее в задачах, где важна максимальная разница между соответствующими элементами векторов, а L2-расстояние может быть предпочтительнее в задачах, где важна общая длина векторов. Косинусное расстояние часто используется в задачах классификации текстов.

### 1.3.3 Реализация в Python

Реализация основных операций с векторами рассмотрена в jupyter notebook файле: lecture\_01\_code\_labs\_01\_vectors.ipynb

### 1.3.4 Вопросы для самопроверки

Какое из определений вектора НЕ ЯВЛЯЕТСЯ корректным?

1. Вектор - это элемент линейного пространства, который может быть описан с помощью набора координат.
2. Вектор - это упорядоченный набор чисел, который может быть использован для представления физических величин, таких как направление, скорость, ускорение и т.д.
3. Вектор - это математическая величина, которая измеряет скорость изменения функции по отношению к ее аргументу.
4. Вектор - это объект, который используется для описания направления и длины в пространстве.

Правильные ответы: 3

Выберите правильную формулу для вычисления скалярного произведения векторов v1 и v2 в трехмерном пространстве:

1.  $a \cdot b = a_1 + b_1 + a_2 + b_2 + a_3 + b_3$
2.  $a \cdot b = a_1 * b_1 - a_2 * b_2 - a_3 * b_3$
3.  $a \cdot b = a_1 * b_1 + a_2 * b_2 + a_3 * b_3$
4.  $a \cdot b = a_1 + a_2 + a_3 - b_1 - b_2 - b_3$

Правильные ответы: 3

Выберите правильный ответ для вычисления L2-нормы между векторами v1 (1,2,2) и v2 (1,5,6) в трехмерном пространстве:

1. L2-норма = 3
2. L2-норма = 4
3. L2-норма = 5
4. L2-норма = 6

Правильные ответы: 3

### 1.3.5 Резюме по разделу

В этом разделе мы поговорили о том, что такое векторы.

Вектор в математике - это объект, который имеет определенную длину и направление в пространстве.

Вектор может быть представлен в виде упорядоченного набора чисел, называемых компонентами вектора.

Свойства векторов включают длину, направление, сумму и разность векторов, скалярное произведение, векторное произведение и угол между векторами.

## 1.4 Линейная алгебра: матрицы

### 1.4.1 Определение матрицы

Матрица - это прямоугольная таблица чисел, символов или выражений, разделенных на строки и столбцы. В машинном обучении термины «таблица» и «матрица» могут использоваться взаимозаменяющими, поскольку матрица - это таблица чисел, упорядоченных в строках и столбцах. В матрицах данные часто представляются в виде числовых значений, которые могут использоваться для обучения алгоритмов машинного обучения. Пример матрицы приведен ниже (Таблица 1.1):

	Размер	Цвет	Уши	Бивень
Слон	Очень большой	Серый	Круглые	Два
Носорог	Большой	Серый	Овальные	Один
Кот	Маленький	Любой	Треугольные	Нет

Таблица 1.1: Пример: матрица A

Матрица может быть представлена как набор векторов-строк или векторов-столбцов, каждый из которых содержит элементы матрицы.

Матрица обычно обозначается заглавными буквами латинского алфавита, например, A, B или C. Размеры матрицы называются её размерностью и указывается в виде  $m \times n$ , где m - количество строк, а n - количество столбцов. Формальная запись для матрицы так:  $A^{m \times n} = (a_{ij}) \in \mathbb{R}^{m \times n}$ . Например, размерность матрицы 1.1 равна  $A^{3 \times 4}$  (3 объекта  $\times$  4 признака).

Каждый элемент матрицы может быть обозначен индексами  $(i, j)$ , где i - номер строки, а j - номер столбца. Например, элемент матрицы  $A_{2,4}$  равен «Один».

Обычно для обозначения матриц используются квадратные скобки, но возможно также другие скобки и вертикальные линии:

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{bmatrix} = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{pmatrix} = \left\{ \begin{array}{l} a_{11} \quad a_{12} \\ a_{21} \quad a_{22} \\ a_{31} \quad a_{32} \end{array} \right\} = \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{vmatrix} = \begin{Vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \\ a_{31} & a_{32} \end{Vmatrix}$$

Матрицы в машинном обучении используются для представления наборов данных, где каждая строка представляет отдельный пример или объект, а каждый столбец - отдельный признак или характеристику, а также для обучения моделей. Матрицы могут быть обработаны и использованы для обучения различных моделей машинного обучения, таких как линейная регрессия, деревья решений, метод опорных векторов, нейронные сети и многие другие (Рисунок 1.11). При обучении моделей машинного обучения матрицы используются для поиска закономерностей и корреляций между признаками, которые могут быть использованы для прогнозирования или классификации новых данных. В машинном обучении используются различные операции над матрицами, такие как умножение матриц, транспонирование, инверсия, сингулярное разложение, вычисление ковариационной матрицы и другие. Матрицы также используются для представления различных преобразований данных, таких как нормализация, стандартизация и сжатие данных.

На рисунке 1.11 показан пример использования матриц в машинном обучении. Обрабатываемое изображение превращается в матрицу, если изображение одноканальное, или тензор

(многомерную матрицу), если в изображении более одного цветового канала. Далее из изображения извлекаются признаки (также с помощью матриц), после чего модель может делать предсказания.

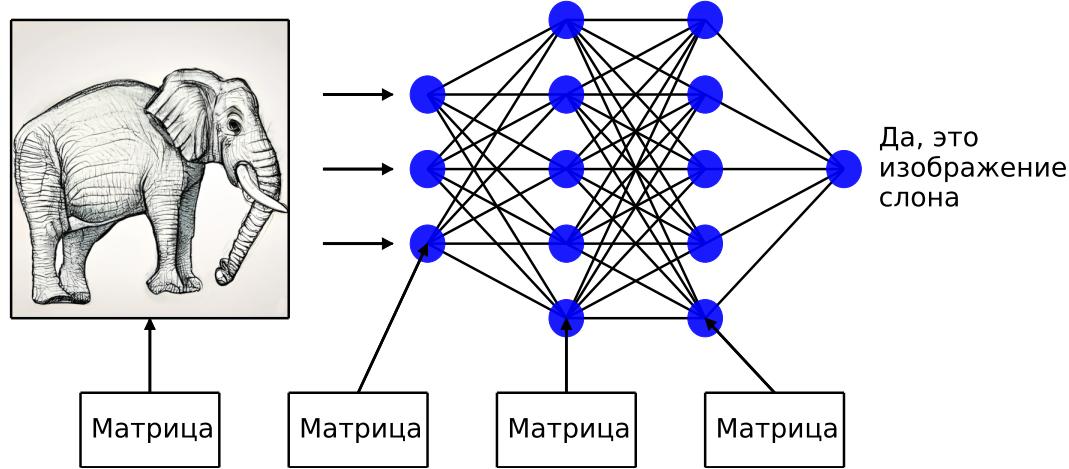


Рисунок 1.11: Матрицы — основа алгоритмов ИИ

## 1.4.2 Операции над матрицами

### Сложение и вычитание матриц

Чтобы сложить или вычесть две матрицы А и В, они должны иметь одинаковое количество строк и столбцов. Если бы мы имели дело с таблицами, мы бы сказали, что все ячейки попарно складываются друг с другом. Если говорить строго, то к элементу матрицы А прибавляется или вычитается соответствующий элемент матрицы В:

$$A \pm B = \begin{bmatrix} a_{11} & \dots & a_{1n} \\ a_{21} & \dots & a_{2n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \dots & a_{mn} \end{bmatrix} \pm \begin{bmatrix} b_{11} & \dots & b_{1n} \\ b_{21} & \dots & b_{2n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \dots & b_{mn} \end{bmatrix} = \begin{bmatrix} a_{11} \pm b_{11} & \dots & a_{1n} \pm b_{1n} \\ a_{21} \pm b_{21} & \dots & a_{2n} \pm b_{2n} \\ \vdots & \ddots & \vdots \\ a_{m1} \pm b_{m1} & \dots & a_{mn} \pm b_{mn} \end{bmatrix}$$

### Умножение матрицы на константу

Чтобы выполнить умножение матрицы А на число с, нужно умножить каждый элемент матрицы А на число с. Результатом будет новая матрица.

$$c \cdot A = c \cdot \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix} = \begin{bmatrix} c \cdot a_{11} & c \cdot a_{12} & \dots & c \cdot a_{1n} \\ c \cdot a_{21} & c \cdot a_{22} & \dots & c \cdot a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ c \cdot a_{m1} & c \cdot a_{m2} & \dots & c \cdot a_{mn} \end{bmatrix}$$

## Транспонирование матрицы

Транспонирование матрицы - это операция, при которой строки матрицы становятся ее столбцами, а столбцы - строками, то есть матрица отражается относительно ее главной диагонали<sup>1</sup>. Обозначение для транспонированной матрицы - символ Т в верхнем правом углу:

$$A = \begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix} \quad A^T = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix}$$

## Произведение матриц

Умножение матрицы на матрицу - это одна из основных операций линейной алгебры. Умножать матрицы можно только в том случае, когда вторая размерность первой матрицы равна первой размерности второй матрицы. Для того, чтобы умножить матрицу А размерности  $m \times n$  на матрицу В размерности  $n \times p$ , необходимо выполнить следующие действия:

1. Создать новую матрицу С, которая будет содержать результат умножения матриц А и В. Новая матрица должна иметь размерность  $m \times p$  (первая размерность первой матрицы  $\times$  вторая размерность второй матрицы)
2. Для каждого элемента матрицы  $C_{i,j}$  вычислить его значение как сумму произведений элементов  $i$ -й строки матрицы А на соответствующие элементы  $j$ -го столбца матрицы В:  $C_{i,j} = A_{i,1} * B_{1,j} + A_{i,2} * B_{2,j} + \dots + A_{i,n} * B_{n,j}$

Для того, чтобы запомнить, как происходит умножение матрицы на матрицу, можно воспользоваться алгоритмом, изображенным на рисунке 1.12. Первую матрицу (матрица А) необходимо поместить слева от итоговой (матрица С), вторую (матрица В) — сверху.

## Использование единичной и обратной матриц

Единичная матрица - это квадратная матрица, у которой на главной диагонали стоят единицы, а остальные элементы равны нулю. Обычно единичную матрицу обозначают буквой I:

$$I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad \text{или} \quad I_{3 \times 3}$$

Обратная матрица - это матрица, умножение которой на исходную матрицу дает единичную матрицу. Обратную матрицу можно обозначить с помощью символа  $-1$  в верхнем правом углу матрицы. Пример исходной, обратной и единичной матриц представлен ниже (Уравнение 1.9):

$$A = \begin{bmatrix} 2 & 5 & 7 \\ 6 & 3 & 4 \\ 5 & -2 & -3 \end{bmatrix} \quad A^{-1} = \begin{bmatrix} 1 & -1 & 1 \\ -38 & 41 & -34 \\ 27 & -29 & 24 \end{bmatrix} \quad I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (1.9)$$

---

<sup>1</sup>Главная диагональ матрицы — это последовательность элементов матрицы, которые расположены на диагонали, идущей от верхнего левого угла матрицы. Главная диагональ состоит из элементов, расположенных на позициях (1,1), (2,2), (3,3) и т.д.

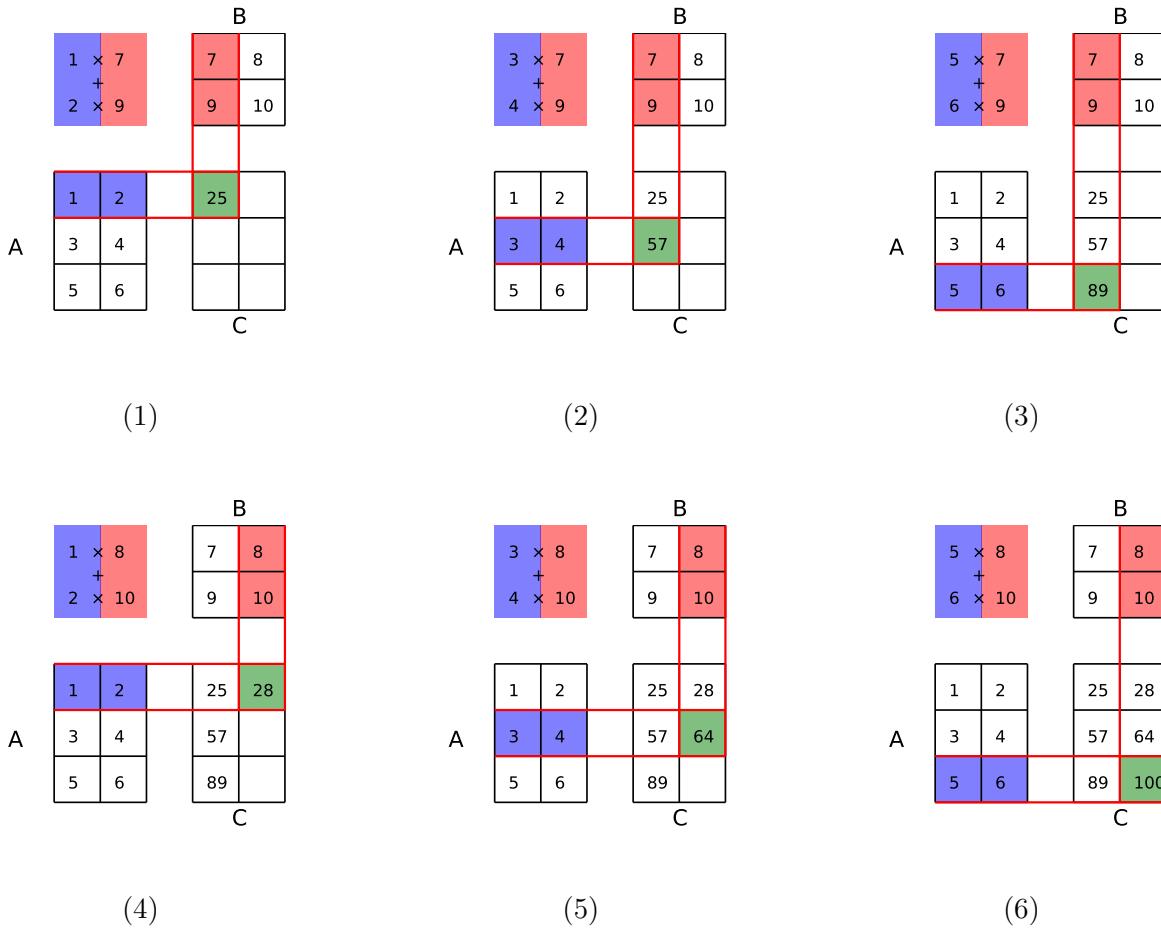


Рисунок 1.12: Визуализация процесса умножения матрицы на матрицу

Единичная и обратная матрицы играют важную роль в линейной алгебре, которая является фундаментальной для многих методов машинного обучения. Единичная матрица используется, например, при решении систем линейных уравнений. Обратная матрица также используется при решении задач оптимизации и настройки параметров моделей машинного обучения. Однако, стоит отметить, что обратная матрица не всегда может быть вычислена. Например, если определитель (мы обсудим, что это, ниже) матрицы равен нулю, то обратной матрицы не существует. Это может возникнуть, например, когда количество признаков превышает количество наблюдений в выборке. Поэтому важно тщательно проверять матрицы на обратимость и использовать методы, которые не требуют обращения матриц, если это возможно.

## Выявление мультиколлинеарности в данных

Мультиколлинеарность - это явление в статистике и анализе данных, когда **две или более переменных в модели сильно коррелируют друг с другом**.

Простейший пример: в наборе данных присутствуют два признака, один из которых описывает расстояние в метрах, другой - в километрах (Таблица 1.2):

	Расстояние до Москвы, км	Расстояние до Москвы, м	Население	Площадь, км <sup>2</sup>
Воронеж	466	466 000	1 057 000	596
Саратов	725	725 000	901 000	393
Тамбов	419	419 000	290 000	107

Таблица 1.2: Пример мультиколлинеарности в данных: расстояние в метрах и километрах линейно зависимы

Эти признаки линейно связаны друг с другом, так что невозможно однозначно определить влияние каждой переменной на целевую переменную. Матрица, содержащая линейно зависимые строки, называется сингулярной или вырожденной. Матрица, не содержащая линейно зависимых строк, называется невырожденной или обратимой.

Линейная зависимость признаков в машинном обучении порождает несколько проблем:

- Переопределение модели: если признаки линейно зависимы, то можно построить множество моделей, которые дадут одинаковый результат. Это может привести к тому, что модель будет слишком сложной и переобученной, т.е. она будет хорошо работать на обучающих данных, но плохо на новых данных.
- Низкая устойчивость: если признаки линейно зависимы, то модель может быть чувствительна к небольшим изменениям входных данных. Например, если один признак можно выразить через другой с небольшой ошибкой, то небольшое изменение входных данных может привести к значительным изменениям в результатах.
- Плохая интерпретируемость: если признаки линейно зависимы, то сложно понять, какой признак вносит больший вклад в результаты модели. Это может быть проблематично для анализа результатов и принятия решений на основе модели.

Мультиколлинеарность может привести к неправильной оценке коэффициентов модели и снижению точности прогнозирования. Кроме того, она может сделать модель менее интерпретируемой, что затрудняет понимание вклада каждой переменной в модель. Если мультиколлинеарность обнаруживается, можно принять меры, такие как исключение одной из коррелирующих переменных из модели, объединение коррелирующих переменных в одну переменную или использование регуляризации для уменьшения влияния некоторых переменных на модель.

Для выявления мультиколлинеарности можно использовать статистические методы, такие как корреляционный анализ или метод главных компонент. Далее мы рассмотрим такие методы, как вычисление **ранга** и **определителя** матрицы для выявления мультиколлинеарности в данных. Если матрица признаков имеет низкий ранг или близкий к нулю определитель, это может указывать на наличие мультиколлинеарности в данных.

### Ранг матрицы

Ранг матрицы - это количество линейно независимых строк или столбцов в матрице. Если матрица имеет ранг, равный меньшему из числа строк или столбцов, то она называется вырожденной.

Высокий ранг матрицы признаков означает, что в данных мало корреляции между признаками и это может быть признаком отсутствия мультиколлинеарности.

Одной из интерпретаций ранга матрицы является **мера неизбыточной информации**, которая хранится в матрице. Если строка/столбец выражается как линейная комбинация других строк/столбцов, то информацию в строке/столбце можно считать избыточной, соответственно, ранг матрицы снижается на 1.

Рассмотрим это на простом наглядном примере. Допустим, у нас есть три матрицы (Рисунок 1.13):

Матрица 1	Матрица 2	Матрица 3
 Банан желтый	 Банан желтый	 Банан
 Апельсин оранжевый	 Банан желтый	 Банан
Две единицы информации ранг=2	Одна единица информации ранг=1	Ноль единиц информации ранг=0

Рисунок 1.13: Вычисление определителя для матрицы  $2 \times 2$

Информация в первой матрице:

- строка 1: банан желтый
- строка 2: апельсин оранжевый

Информация во второй матрице:

- строка 1: банан желтый
- строка 2: банан желтый

Информация в третьей матрице:

- строка 1: банан
- строка 2: бана

Мы видим, что в первой матрице содержится информация о двух фактах: цвете банана и апельсина. Соответственно, ранг такой системы будет 2. Ранг второй системы будет равен 1. Третья матрица не хранит никакой информации о цвете фруктов, соответственно, её ранг равен 0.

Рассмотрим другой пример, с числовыми значениями(Рисунок 1.14):

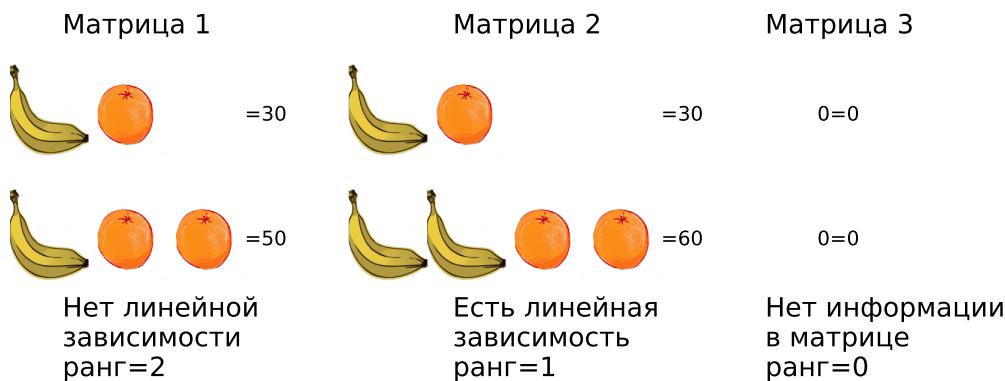


Рисунок 1.14: Вычисление определителя для матрицы  $2 \times 2$

- строка 1: 1 банан и 1 апельсин стоят 30 рублей
- строка 2: 1 банан и 2 апельсина стоят 50 рублей

Информация во второй матрице:

- строка 1: 1 банан и 1 апельсин стоят 30 рублей
- строка 2: 2 банана и 2 апельсина стоят 60 рублей

Информация в третьей матрице:

- строка 1: 0 бананов и 0 апельсинов стоят 0 рублей
- строка 2: 0 бананов и 0 апельсинов стоят 0 рублей

Если рассматривать матрицы как системы линейных уравнений, мы видим что в первой матрице нет линейной зависимости строк, и имеется единственное решение: 1 банан стоит 10 рублей и 1 апельсин стоит 20 рублей. Ранг матрицы равен 2. Вторая матрица содержит линейно зависимые строки (строка 2 равна строке 1 $\times$ 2), соответственно, решение системы уравнений сводится к линии. Ранг матрицы равен 1. Третья матрица не содержит никакой информации, её решение сводится к плоскости. Ранг матрицы равен 0.

Существует несколько методов вычисления ранга матрицы:

- Метод Гаусса-Жордана: матрица приводится к ступенчатому виду, затем вычисляется количество ненулевых строк, которое и является рангом матрицы
- Метод элементарных преобразований: матрица приводится к эквивалентной матрице, которая имеет минимальный набор строк и столбцов, образующих базис в пространстве строк (или столбцов) матрицы. Количество таких строк (или столбцов) и будет равно рангу матрицы
- Метод определителей: ранг матрицы равен максимальному порядку ненулевых миноров матрицы
- Метод сингулярного разложения: ранг матрицы равен количеству ненулевых сингулярных значений матрицы
- Метод Грама-Шмидта: матрица приводится к ортонормированной матрице, затем вычисляется количество ненулевых строк, которое и является рангом матрицы

Мы не будем разбирать их подробно (это делается в полноценных курсах по линейной алгебре). Для вычисления ранга матриц мы будем пользоваться библиотекой numpy.

### Определитель матрицы

Определитель матрицы (его еще называют детерминантом) - это число, которое вычисляется из элементов **квадратной** матрицы и используется для определения, является ли матрица вырожденной или нет.

Определитель, близкий к нулю, указывает на наличие мультиколлинеарности в данных.

Рассмотрим понятие определителя на двух простых наглядных примерах с матрицами размерностью  $2 \times 2$  и  $3 \times 3$ .

Допустим у нас есть матрица A:

$$A = \begin{bmatrix} a & b \\ c & d \end{bmatrix}$$

Определитель будет вычисляться по формуле:  $a \times d - b \times c$  (произведение элементов на главной диагонали матрицы минус произведение элементов на побочной диагонали<sup>1</sup> матрицы). Если он равен 0, то строки линейно зависимы.

Давайте рассмотрим, почему это так (Рисунок 1.15). Если мы допускаем, что строки линейно зависимы, то строка 2 равна строке 1, умноженной на какой-то коэффициент k. Тогда:

$$\begin{aligned} a \times k &= c, \quad b \times k = d \Rightarrow \\ k = \frac{c}{a} &= \frac{d}{b} \Rightarrow \\ a \times d &= b \times c \Rightarrow \\ a \times d - b \times c &= 0 \end{aligned}$$

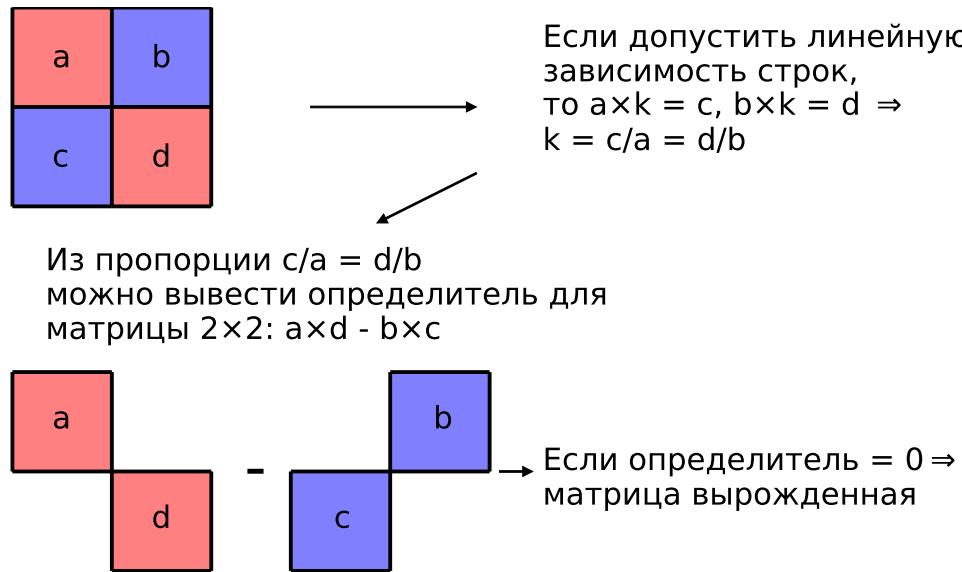
Определитель матрицы  $3 \times 3$  можно вычислить по формуле Саррюса. Пусть дана матрица:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

Тогда определитель можно вычислить следующим образом:

---

<sup>1</sup>Побочная диагональ матрицы — это последовательность элементов матрицы, которые расположены на диагонали, идущей от верхнего правого угла матрицы. Побочная диагональ состоит из элементов, расположенных на позициях  $(1,n)$ ,  $(2,n-1)$ ,  $(3,n-2)$  и т.д., где n - размерность матрицы.

Рисунок 1.15: Вычисление определителя для матрицы  $2 \times 2$ 

- Создать два вспомогательных столбца путем копирования первых двух столбцов матрицы за третьим столбцом
- Перемножить элементы по диагоналям, идущим слева направо, и сложить полученные произведения:  $a_{11} * a_{22} * a_{33} + a_{12} * a_{23} * a_{31} + a_{13} * a_{21} * a_{32}$
- Перемножить элементы по диагоналям, идущим справа налево, и вычесть полученные произведения:  $-a_{13} * a_{22} * a_{31} - a_{11} * a_{23} * a_{32} - a_{12} * a_{21} * a_{33}$

Результатом будет определитель матрицы:  $\det = a_{11} * a_{22} * a_{33} + a_{12} * a_{23} * a_{31} + a_{13} * a_{21} * a_{32} - a_{13} * a_{22} * a_{31} - a_{11} * a_{23} * a_{32} - a_{12} * a_{21} * a_{33}$

Иллюстрация формулы Саррюса приведена на рисунке 1.16.

### 1.4.3 Реализация в Python

Реализация основных операций с матрицами рассмотрена в jupyter notebook файле: lecture\_01\_code\_labs\_02\_matrices.ipynb

На этом мы заканчиваем знакомство с основными инструментами линейной алгебры, применяемыми в машинном обучении. Давайте перейдем к математическому анализу.

### 1.4.4 Вопросы для самопроверки

Какие из определений матрицы являются корректным?

- Матрица - это прямоугольная таблица чисел, разбитая на строки и столбцы.

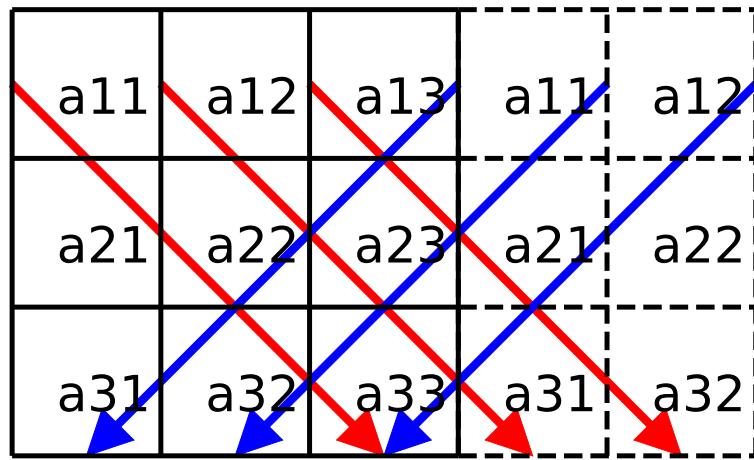


Рисунок 1.16: Иллюстрация формулы Саррюса

2. Матрица - это математический объект, который используется в машинном обучении для представления наборов данных.
3. Матрица - это математический объект, который используется в машинном обучении для сохранения признаков объектов.
4. Матрица - это последовательность шагов алгоритма машинного обучения.

Правильные ответы: 1,2,3

Транспонирование матрицы - это ...:

1. операция, при которой строки матрицы становятся ее столбцами, а столбцы - строками, матрица отражается относительно ее главной диагонали
2. операция, при которой матрица отражается зеркально по горизонтальной оси
3. операция, при которой матрица отражается зеркально по вертикальной оси
4. операция, при которой в матрицу добавляются новые строки/столбцы

Правильные ответы: 1

Произведение матриц А @ В с какими размерностями допустимо:

1. размерность А = 3x3, размерность В = 5x3
2. размерность А = 5x3, размерность В = 3x5
3. размерность А = 5x5, размерность В = 3x3
4. размерность А = 5x3, размерность В = 5x3

Правильные ответы: 2

#### 1.4.5 Резюме по разделу

В этом разделе мы рассмотрели матрицы. Матрица в машинном обучении - это прямоугольный массив чисел, который представляет собой таблицу, состоящую из строк и столбцов. Некоторые из основных свойств матриц:

- Размерность: количество строк и столбцов, которые определяют размерность матрицы.
- Операции: матрицы можно складывать, вычитать, умножать на число и на другие матрицы.
- Транспонирование: матрица может быть транспонирована, что означает замену строк на столбцы и наоборот.
- Обратная матрица: некоторые матрицы имеют обратную матрицу, которая позволяет решить систему линейных уравнений.
- Вырожденная матрица - это квадратная матрица, у которой определитель равен нулю.
- Определитель матрицы - это число, которое вычисляется из элементов квадратной матрицы и используется для решения линейных систем уравнений, нахождения обратной матрицы, определения линейной зависимости векторов и других задач линейной алгебры.
- Ранг матрицы: максимальное число линейно независимых строк или столбцов в матрице. Ранг матрицы и ее вырожденность тесно связаны. Если матрица вырождена, то ее определитель равен нулю. Следовательно, ранг вырожденной матрицы будет меньше, чем ее размерность.

Матрицы широко используются в машинном обучении, особенно в задачах обработки данных. Например, в задаче классификации изображений матрица пикселей может быть использована для обучения модели.

## 1.5 Математический анализ: функции

### 1.5.1 Определение функции

В математике функция - это соответствие между двумя множествами, где каждому элементу из одного множества (называемого «область определения» (англ. Domain of a function)) сопоставляется единственный элемент из другого множества (называемого «область значений» (англ. Range of a function)).

**Область определения** функции — это множество всех значений аргумента (переменной  $x$ ).

**Область значений** функции - это множество всех значений функции (переменной  $y$ ). Функция обозначается обычно символом  $f$  и записывается в виде  $f(x)$ , где  $x$  - элемент из области определения функции. Область определений и область значений функции могут быть любыми множествами, в том числе и множествами действительных чисел, комплексных чисел, векторов и т.д. Обычно, функции в математическом анализе отображают элементы из множества вещественных чисел на другие вещественные числа. Функции используются для описания различных зависимостей и отношений между различными величинами, например, для определения зависимости скорости от времени, температуры от высоты, или для расчета значения функции в определенной точке.

Функции могут быть заданы различными способами, например, аналитически, графически или таблично.

**Аналитически заданная функция** представляет собой формулу, которая позволяет вычислить значение функции для любого заданного значения аргумента.

**Графически заданная функция** представляет собой кривую, которая отображает значения функции на оси координат.

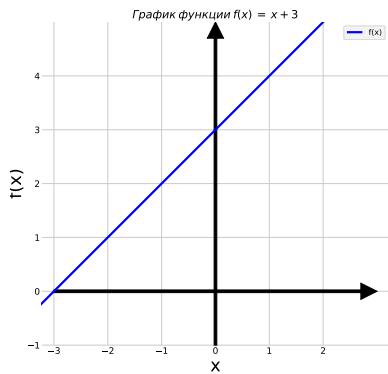
**Таблично заданная функция** представляет собой список пар значений, где первый элемент пары является аргументом функции, а второй - значение функции.

Функции в математическом анализе играют важную роль в решении различных задач, таких как оптимизация, нахождение экстремумов, решение дифференциальных уравнений и т.д. Они также являются основой для изучения других математических дисциплин, таких как теория вероятностей и статистика.

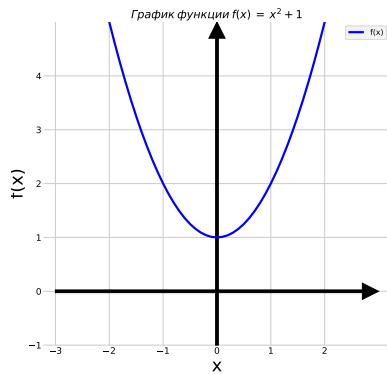
### 1.5.2 Свойства функций

В математике функции обладают многими свойствами, некоторые из которых важны для их понимания и использования. Ниже перечислены некоторые из основных свойств функций:

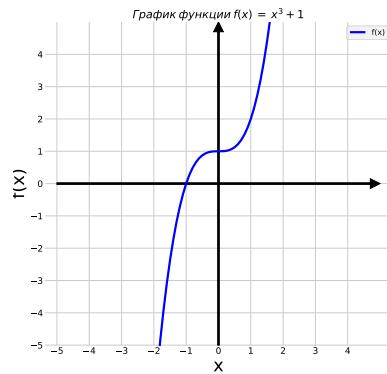
- Однозначность: каждому элементу из области определения функции соответствует единственный элемент из области значений функции.
- Непрерывность: функция называется непрерывной, если изменение значения функции в результате бесконечно малого изменения аргумента также является бесконечно малым.
- Гладкость: функция называется гладкой, если ее производные любого порядка непрерывны.
- Ограниченнность: функция называется ограниченной, если ее значения на области определения ограничены константой.
- Монотонность: функция называется монотонной, если она убывает или возрастает на всей области определения или на некотором ее интервале.



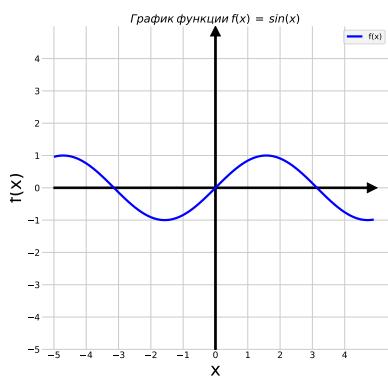
(1) Однозначность функции



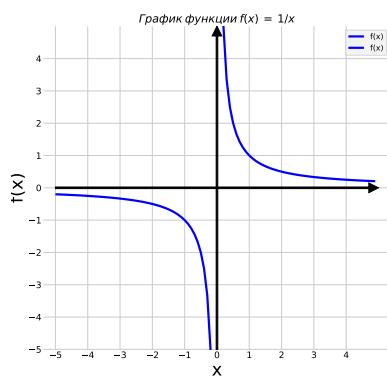
(2) Непрерывность функции



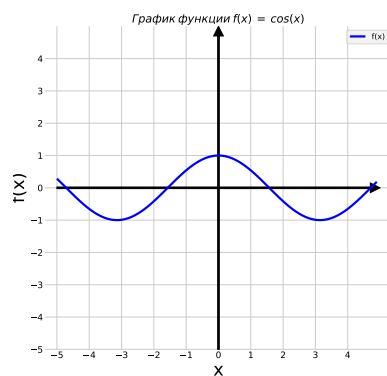
(3) Гладкость функции



(4) Ограниченнность функции



(5) Монотонность функции



(6) Периодичность функции

Рисунок 1.17: Иллюстрация основных свойств функций

- Периодичность: функция называется периодической, если ее значения повторяются через определенный интервал.

Приведем иллюстрацию описанных свойств на примере распространенных функций (Рисунок 1.17).

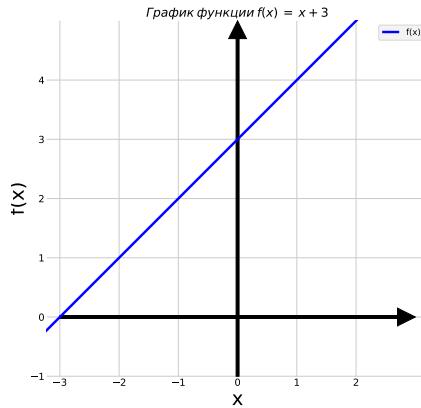
### 1.5.3 Графики часто используемых функций

$$f(x) = ax + b$$

График функции  $f(x) = ax + b$  при  $a = 1$  и  $b = 3$  (Рисунок 1.18):

Свойства функции  $y = ax + b$ :

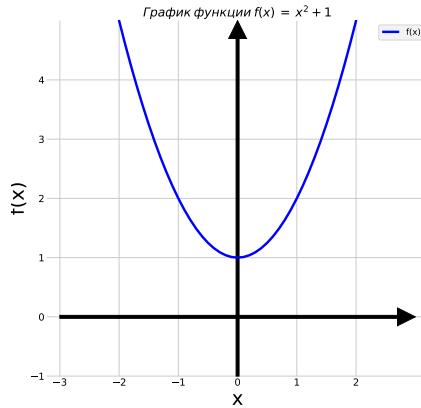
- Гладкость: функция  $y = ax + b$  гладкая на всей области определения, т.е. не имеет разрывов, углов и изломов.
- Ограниченнность: функция  $y = ax + b$  не является ограниченной сверху или снизу на всей области определения, за исключением случая, когда  $a = 0$ , тогда функция будет ограничена только снизу, если  $b < 0$ , или ограничена только сверху, если  $b > 0$ .

Рисунок 1.18:  $f(x) = ax + b$ 

- Монотонность: функция  $y = ax + b$  может быть монотонной или не монотонной, в зависимости от знака коэффициента  $a$ . Если  $a > 0$ , то функция возрастает, если  $a < 0$ , то функция убывает, а если  $a = 0$ , то функция константа и не является монотонной.
- Периодичность: функция  $y = ax + b$  не является периодической на всей области определения, за исключением случая, когда  $a = 0$ , тогда функция является периодической с любым периодом, проходящим через начало координат.

$$f(x) = ax^2 + b$$

График функции  $f(x) = ax^2 + b$  при  $a = 1$  и  $b = 1$  (Рисунок 1.19):

Рисунок 1.19:  $f(x) = ax^2 + b$ 

Свойства функции  $y = ax^2 + b$ :

- Гладкость: функция  $y = ax^2 + b$  гладкая на всей области определения, т.е. имеет непрерывные производные любого порядка.
- Ограниченностъ: функция  $y = ax^2 + b$  может быть ограниченной сверху или снизу на всей области определения, в зависимости от знака коэффициента  $a$  и значения  $b$ . Если  $a > 0$ , то функция ограничена снизу значением  $b$ , а сверху не ограничена. Если  $a <$

0, то функция ограничена сверху значением  $b$ , а снизу не ограничена. Если  $a = 0$ , то функция является константой и ограничена сверху и снизу значением  $b$ .

- Монотонность: функция  $y = ax^3 + b$  не является монотонной на всей области определения.
- Периодичность: функция  $y = ax^3 + b$  не является периодической на всей области определения.

$$f(x) = ax^3 + b$$

График функции  $f(x) = ax^3 + b$  при  $a = 1$  и  $b = 1$  (Рисунок 1.20):

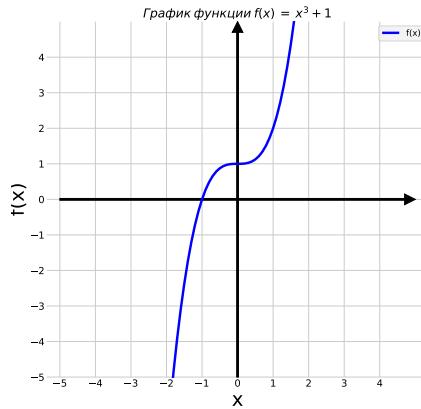


Рисунок 1.20:  $f(x) = ax^3 + b$

Свойства функции  $y = ax^3 + b$ :

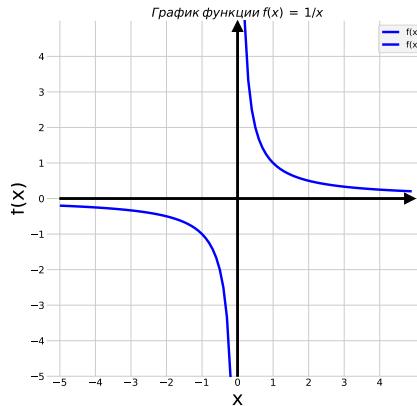
- Гладкость: функция  $y = ax^3 + b$  гладкая на всей области определения, т.е. имеет непрерывные производные любого порядка.
- Ограниченнность: функция  $y = ax^3 + b$  может быть ограниченной сверху или снизу на всей области определения, в зависимости от знака коэффициента  $a$  и значения  $b$ . Если  $a > 0$ , то функция ограничена снизу значением  $b$ , а сверху не ограничена. Если  $a < 0$ , то функция ограничена сверху значением  $b$ , а снизу не ограничена. Если  $a = 0$ , то функция является константой и ограничена сверху и снизу значением  $b$ .
- Монотонность: функция  $y = ax^3 + b$  может быть монотонной или не монотонной, в зависимости от знака коэффициента  $a$ . Если  $a > 0$ , то функция возрастает на всей области определения, если  $a < 0$ , то функция убывает на всей области определения, а если  $a = 0$ , то функция константа и не является монотонной.
- Периодичность: функция  $y = ax^3 + b$  не является периодической на всей области определения.

$$f(x) = 1/x$$

График функции  $f(x) = 1/x$  (Рисунок 1.21):

Свойства функции  $f(x) = 1/x$ :

- Гладкость: функция  $f(x) = 1/x$  не является гладкой на всей области определения, так как имеет вертикальную асимптоту в точке  $x=0$ , где она не определена. Однако, на

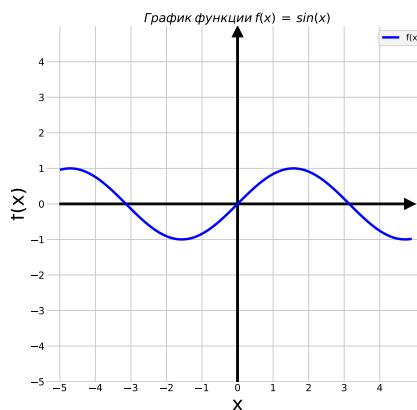
Рисунок 1.21:  $f(x) = 1/x$ 

любом интервале, не содержащем точку  $x=0$ , функция гладкая и имеет непрерывные производные любого порядка.

- Ограничность: функция  $f(x) = 1/x$  не является ограниченной на своей области определения, так как ее значения могут стать очень большими, если  $x$  близко к 0. Например, при  $x=0.0001$ , значение функции будет равно 10 000, а при  $x=0.00001$  - 100 000. Таким образом, функция не имеет ограничений ни сверху, ни снизу.
- Монотонность: функция  $f(x) = 1/x$  является монотонно убывающей на своей области определения, так как при увеличении аргумента  $x$ , ее значение убывает. Формально, для любых двух точек  $x_1$  и  $x_2$  на своей области определения, если  $x_1 < x_2$ , то  $f(x_1) > f(x_2)$ .
- Периодичность:  $f(x) = 1/x$  не является периодической на своей области определения, так как ее значения зависят только от значения аргумента  $x$  и не повторяются через равные интервалы.

$$f(x) = \sin(x)$$

График функции  $(x) = \sin(x)$  (Рисунок 1.22):

Рисунок 1.22:  $(x) = \sin(x)$

Свойства функции  $f(x) = \sin(x)$ :

- Гладкость: функция  $\sin(x)$  бесконечно дифференцируема на всей числовой оси. Это означает, что ее производные любого порядка существуют везде на числовой оси и непрерывны.
- Ограничность: функция  $\sin(x)$  ограничена сверху и снизу значениями от -1 до 1. Таким образом, ее значения не могут превышать этих границ.
- Монотонность: функция  $\sin(x)$  не является монотонной на всей числовой оси, поскольку имеет периодическую структуру. Однако на каждом периоде функция  $\sin(x)$  является ограниченной и возрастающей на первой половине периода и убывающей на второй половине периода.
- Периодичность: функция  $\sin(x)$  периодическая с периодом  $2\pi$ . Это означает, что для любого  $x$  значение функции  $\sin(x)$  будет равно значению функции  $\sin(x+2\pi)$ ,  $\sin(x+4\pi)$ ,  $\sin(x-2\pi)$  и т.д.

$$f(x) = \cos(x)$$

График функции  $f(x) = \cos(x)$  (Рисунок 1.23):

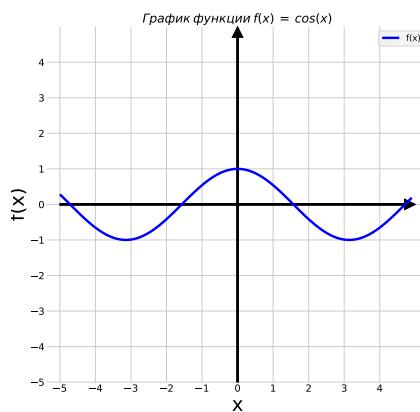


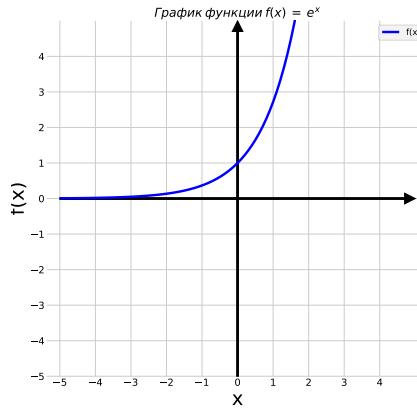
Рисунок 1.23:  $f(x) = \cos(x)$

Свойства функции  $f(x) = \cos(x)$ :

- Гладкость: функция  $\cos(x)$  является бесконечно дифференцируемой на всей числовой оси, что означает, что она гладкая на любом интервале.
- Ограничность: функция  $\cos(x)$  ограничена сверху и снизу значениями от -1 до 1, т.е.  $|\cos(x)| \leq 1$  для любого  $x$ .
- Монотонность: функция  $\cos(x)$  не является монотонной на всей числовой оси, так как она пересекает ось  $x$  бесконечное число раз. Однако на каждом периоде функция  $\cos(x)$  монотонно убывает с увеличением  $x$  на первой половине периода, и монотонно возрастает на второй половине.
- Периодичность: функция  $\cos(x)$  является периодической с периодом  $2\pi$ , т.е.  $\cos(x + 2\pi) = \cos(x)$  для любого  $x$ .

$$f(x) = e^x$$

График функции  $f(x) = e^x$  (Рисунок 1.24):

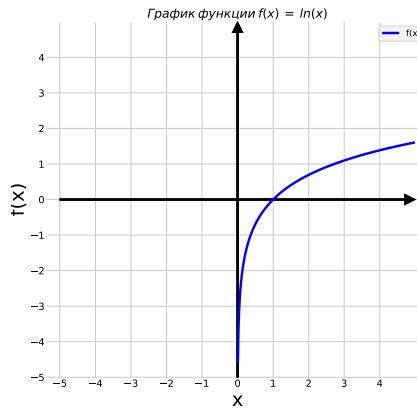
Рисунок 1.24:  $f(x) = e^x$ 

Свойства функции  $f(x) = e^x$ :

- Гладкость: функция  $e^x$  является бесконечно дифференцируемой на всей числовой оси, что означает, что она гладкая на любом интервале.
- Ограниченнность: функция  $e^x$  неограничена сверху, но ограничена снизу нулем, т.е.  $e^x > 0$  для любого x.
- Монотонность: функция  $e^x$  монотонно возрастает на всей числовой оси. Это означает, что с увеличением x значение функции  $e^x$  также увеличивается.
- Периодичность: функция  $e^x$  не является периодической на всей числовой оси.

$$f(x) = \ln(x)$$

График функции  $f(x) = \ln(x)$  (Рисунок 1.25):

Рисунок 1.25:  $f(x) = \ln(x)$ 

Свойства функции  $f(x) = \ln(x)$ :

- Гладкость: функция  $\ln(x)$  гладкая на всей области определения ( $x > 0$ ), то есть она бесконечно дифференцируема.
- Ограниченнность: функция  $\ln(x)$  неограничена сверху, но ограничена снизу нулем, то есть  $\ln(x) > 0$  для  $x > 1$  и  $\ln(1) = 0$ .

- Монотонность: функция  $\ln(x)$  монотонно возрастает на всей области определения, то есть если  $x_1 < x_2$ , то  $\ln(x_1) < \ln(x_2)$ .
- Периодичность: функция  $\ln(x)$  не является периодической на всей числовой оси.

#### 1.5.4 Реализация в Python

Реализация основных функций рассмотрена в jupyter notebook файле: lecture\_01\_code\_- labs\_03\_functions.ipynb

#### 1.5.5 Вопросы для самопроверки

Область значений функции - это ...:

1. множество всех значений аргумента (переменной  $x$ )
2. множество всех значений функции (переменной  $y$ )
3. объединение множества всех значений аргумента (переменной  $x$ ) и множества всех значений функции (переменной  $y$ )
4. пересечение множества всех значений аргумента (переменной  $x$ ) и множества всех значений функции (переменной  $y$ )

Правильные ответы: 2

Функция называется гладкой, если ...

1. изменение значения функции в результате бесконечно малого изменения аргумента также является бесконечно малым
2. ее значения на области определения ограничены константой
3. ее значения повторяются через определенный интервал
4. ее производные любого порядка непрерывны

Правильные ответы: 4

Какие из приведенных ниже функций являются периодическими?

1.  $y = a * x^2 + b$
2.  $y = \sin(x)$
3.  $y = \cos(x)$
4.  $y = 1/x$

Правильные ответы: 2, 3

#### 1.5.6 Резюме по разделу

В этом разделе мы поговорили о функциях. Функция - это правило, которое отображает один набор значений (аргументов) в другой набор значений (значений функции). Функции могут быть определены математически, графически, в виде алгоритмов или программного кода. Некоторые из основных свойств функций:

- Гладкость функции: функция называется гладкой, если ее производные любого порядка непрерывны.

- Ограниченнность: свойство функции, которое описывает, насколько функция ограничена в своих значениях.
- Монотонность: свойство функции, которое описывает ее возрастание или убывание при изменении аргумента.
- Периодичность: свойство функции, которое описывает ее повторение с определенным периодом.
- Дифференцируемость: функция является дифференцируемой, если ее производная существует для всех точек в области определения.

Функции широко используются в машинном обучении, в задачах оптимизации, моделирования и прогнозирования. Например, в линейной регрессии функция используется для описания связи между независимыми переменными и зависимой переменной.

## 1.6 Математический анализ: производные

### 1.6.1 Определение производной

Производная - это понятие, используемое в математике для описания того, как быстро изменяется функция в каждой точке. Если мы говорим о функции, которая описывает зависимость одной переменной от другой, то производная показывает, как быстро изменяется значение функции с изменением ее аргумента в каждой точке. Один из самых часто встречающихся примеров производной в реальной жизни — это ускорение. Ускорение показывает, как быстро изменяется скорость тела за единицу времени.

Более формально, производная функции определяется как предел отношения приращения функции к приращению ее аргумента при стремлении приращения аргумента к нулю:

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

где  $\Delta x$  - бесконечно малое приращение аргумента.

Эта формула показывает, как быстро изменяется функция  $f(x)$  в точке  $x$ . Если производная положительна, то функция возрастает в этой точке, если отрицательна - функция убывает. Если производная равна нулю, то функция имеет экстремум (максимум или минимум) в этой точке.

Производная имеет много приложений в математике и науке, так как она позволяет описать изменение величин в различных процессах и является ключевым инструментом для решения многих задач.

Существует несколько нотаций (способов записи) производной функции. Наиболее распространенные из них:

- Производная в нотации Лейбница:  $\frac{dy}{dx}$ . Здесь  $y$  - это функция от  $x$ , а дробь указывает на приращение функции  $y$  по приращению переменной  $x$ . Если нужно указать производную более высокого порядка, то можно использовать следующую запись:  $\frac{d^n y}{dx^n}$ . Здесь  $n$  указывает на порядок производной.
- Производная в нотации Лагранжа:  $y'(x)$  или  $f'(x)$ . Эта запись эквивалентна нотации Лейбница  $\frac{dy}{dx}$ .
- Дифференциал функции:  $df = f'(x)dx$ . Эта запись означает дифференциал функции  $f(x)$ , который можно представить как произведение производной  $f'(x)$  и бесконечно малого приращения  $dx$ .

Выбор записи производной зависит от предпочтений и требований к форматированию в конкретном случае.

Геометрическая интерпретация производной функции заключается в том, что производная в точке определяет скорость изменения функции в этой точке (Рисунок 1.26). При стремлении приращения аргумента  $x$  (красные пунктирные линии) к нулю отношение приращения функции к приращению аргумента будет все точнее отражать скорость роста функции в точке.

В машинном обучении геометрическая интерпретация производной имеет множество применений. Например, она может использоваться в оптимизации функций потерь, которые определяют, насколько хорошо модель машинного обучения соответствует данным. Методы оптимизации, такие как градиентный спуск, используют производную для нахождения минимума функции потерь и оптимизации параметров модели.

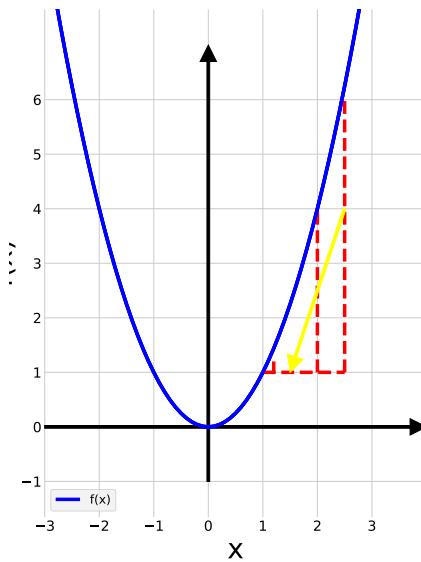


Рисунок 1.26: Геометрическая интерпретация производной

Если мы рассмотрим график функции  $y = f(x)$ , то производная функции  $f'(x)$  в каждой точке графика показывает наклон касательной к этой точке графика. Если производная положительна, то наклон касательной положительный, что означает, что функция в этой точке возрастает. Если производная отрицательна, то наклон касательной отрицательный, что означает, что функция в этой точке убывает. Если производная равна нулю, то касательная к графику функции является горизонтальной и функция имеет экстремум (максимум или минимум) в этой точке (Рисунок 3.3):

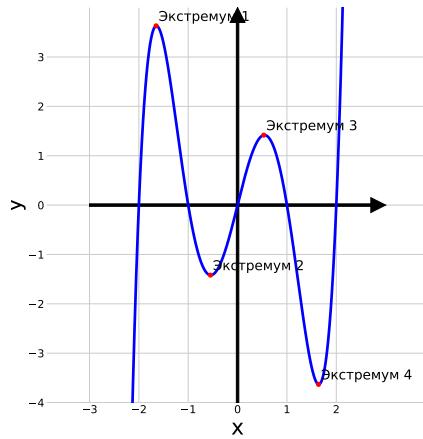


Рисунок 1.27: Нахождение экстремумов - одна из основных задач производных

Задачи машинного обучения связаны с оптимизацией параметров модели, чтобы минимизировать ошибку на тренировочных данных или максимизировать некоторую функцию потерь. Нахождение экстремумов функций помогает найти оптимальные значения этих параметров.

Если график функции  $y = f(x)$  имеет касательную в точке  $(a, f(a))$ , то производная функции в этой точке равна тангенсу угла наклона касательной к графику в этой точке. Пусть  $m$  - угол наклона касательной к графику функции  $y = f(x)$  в точке  $(a, f(a))$ . Тогда, если мы рассмотрим малый прирост аргумента  $dx$  и соответствующий прирост функции  $df$ , то получим следующее:

$$\tan m = \lim_{dx \rightarrow 0} \frac{df}{dx}$$

Если существует производная функции  $f(x)$  в точке  $a$ , то этот предел существует и равен значению производной  $f'(a)$ , т.е.

$$\tan m = \frac{df}{dx}$$

Таким образом, производная функции в точке касания графика касательной к этому графику равна тангенсу угла наклона касательной в этой точке.

## 1.6.2 Недифференцируемые функции

Недифференцируемая функция - это функция, которая не имеет производной в какой-либо точке своей области определения. То есть, производная функции не существует в этой точке, либо существует, но не является конечной.

### Примеры недифференцируемых функций

Любая функция, в которой есть углы, будет недифференцируема в точке нахождения угла. Функция модуля знака  $|x|$  является недифференцируемой в нуле, так как ее производная не существует в этой точке. Если попытаться провести касательную, она будет определена неоднозначно (Рисунок 1.28):

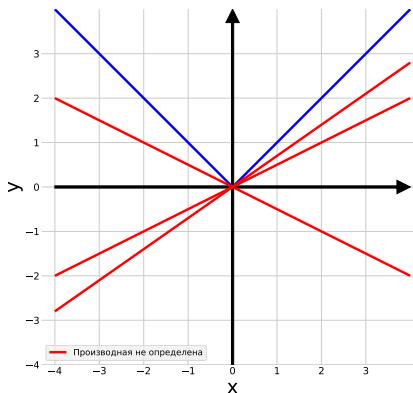


Рисунок 1.28: Функция  $|x|$  недифференцируема в точке 0

Разрывные функции — функции, имеющая разрыв в некоторых точках — также недифференцируемы в этих точках (Рисунок 1.29):

Недифференцируемые функции могут вызывать некоторые проблемы при решении математических задач и приложений. Некоторые из проблем, связанных с недифференцируемыми функциями, включают:

- Ограничение области определения: некоторые задачи требуют дифференцируемых функций, и использование недифференцируемых функций может ограничить область определения их применения.

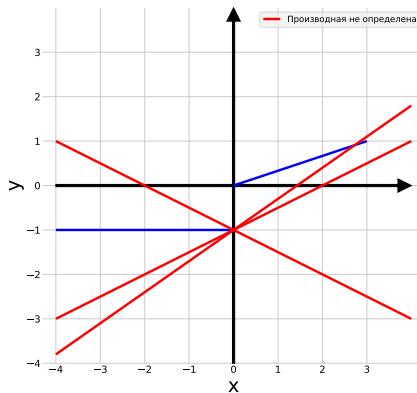


Рисунок 1.29: Разрывная (кусочно-линейная) функция недифференцируема в точке 0

- Сложность вычислений: недифференцируемые функции могут быть более сложными в вычислении, так как их производные могут иметь особенности, такие как разрывы или бесконечные значения.
- Несуществование градиента: в некоторых задачах оптимизации требуется нахождение градиента функции, и недифференцируемые функции могут быть проблемой, так как градиент не существует в некоторых точках.
- Необходимость использования других методов: некоторые задачи могут требовать анализа недифференцируемых функций, и для их решения может потребоваться использование других методов, например, методов интегрального и функционального анализа.

Функции являются недифференцируемыми в точках, где тангенс имеет вертикальные асимптоты. Это происходит потому, что в таких точках производная функции не существует. Например, функция тангенса имеет вертикальные асимптоты в точках  $\frac{\pi}{2} + k\pi$ , где  $k$  - любое целое число. В этих точках тангенс стремится к бесконечности, и производная функции не существует (Рисунок 1.30):

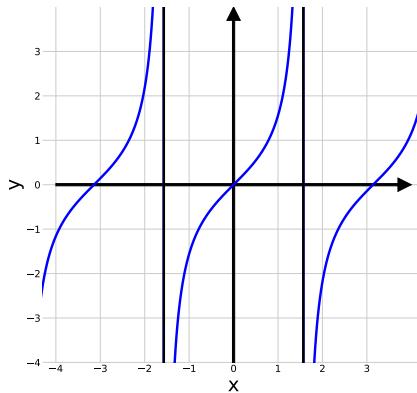


Рисунок 1.30: Разрывная (кусочно-линейная) функция недифференцируема в точке 0

Соответственно, функция является недифференцируемой, если:

- имеются углы

- имеются разрывы
- имеются точки с вертикальными асимптотами тангенса

### 1.6.3 Правила дифференцирования

Основные правила дифференцирования:

- Сумма производных:

$$(f(x) + g(x))' = f'(x) + g'(x)$$

- Разность производных:

$$(f(x) - g(x))' = f'(x) - g'(x)$$

- Произведение производных:

$$(f(x)g(x))' = f'(x)g(x) + f(x)g'(x)$$

- Частное производных:

$$\left(\frac{f(x)}{g(x)}\right)' = \frac{f'(x)g(x) - f(x)g'(x)}{[g(x)]^2}$$

- Производная произведения функции на константу равна произведению константы и производной функции:

$$(cf(x))' = cf'(x)$$

- Производная константы равна нулю:

$$\frac{d}{dx}[f(x) + C] = f'(x)$$

- Правило цепочки: производная композиции функций<sup>1</sup> равна произведению производной внешней функции и производной внутренней функции:

$$(f(g(x)))' = f'(g(x))g'(x)$$

Это правило играет важную роль в машинном обучении. В нотации Лейбница оно может быть записано следующим образом:

$$\frac{df}{dx} = \frac{df}{dg} \cdot \frac{dg}{dx}$$

Эта формула показывает, что производная сложной функции  $f(g(x))$  равна произведению производной внешней функции  $f'(g(x))$  и производной внутренней функции  $g'(x)$ . Например, если  $f(x) = \sin(x^2)$  и  $g(x) = x^2$ , то

$$\frac{df}{dx} = \frac{d}{dx} \sin(g(x)) = \cos(g(x)) \cdot \frac{dg}{dx} = \cos(x^2) \cdot 2x$$

Это правило может быть использовано для дифференцирования более сложных функций, которые представляют собой композицию нескольких функций.

Эти правила могут быть расширены и использованы в более сложных случаях, но они обра- зуют основу дифференцирования функций.

---

<sup>1</sup>Композиция функций, или сложная функция — операция, при которой результат одной функции используется в качестве аргумента для другой функции

### 1.6.4 Производные часто используемых функций

Ниже приведены производные часто используемых функций:

- Константа:

$$f'(C) = 0$$

- Линейная функция:

$$f'(ax + b) = a$$

- Степенная функция:

$$f'(x^n) = nx^{n-1}$$

- Степенная функция с отрицательным целым показателем:

$$f'(x^{-n}) = -nx^{-n-1}$$

- Тригонометрические функции:

$$f'(\sin(x)) = \cos(x)$$

$$f'(\cos(x)) = -\sin(x)$$

$$f'(\tan(x)) = \sec^2(x)$$

Обратные тригонометрические функции:

$$f'(\arcsin(x)) = \frac{1}{\sqrt{1-x^2}}$$

$$f'(\arccos(x)) = -\frac{1}{\sqrt{1-x^2}}$$

$$f'(\arctan(x)) = \frac{1}{1+x^2}$$

- Экспоненциальная функция:

$$f'(e^x) = e^x$$

- Логарифмическая функция:

$$f'(\ln(x)) = \frac{1}{x}$$

Эти производные могут быть использованы для нахождения производных более сложных функций, которые представляют собой композицию нескольких функций.

### 1.6.5 Реализация в Python

Реализация основных производных рассмотрена в jupyter notebook файле: lecture\_01\_-code\_labs\_04\_derivatives.ipynb

### 1.6.6 Вопросы для самопроверки

Какие из перечисленных ниже определений производной являются верными?

1. наклон касательной к графику функции в данной точке.
2. показатель скорости изменения функции в данной точке.
3. отношение значения аргумента к значению функции.
4. отношение изменения функции к изменению ее аргумента при бесконечно малом изменении аргумента.

Правильные ответы: 1, 2, 4

Функция является недифференцируемой, если:

1. имеются точки с вертикальными асимптотами тангенса
2. имеются разрывы
3. имеются углы
4. значения функции на порядки превосходят значения аргумента

Правильные ответы: 1, 2, 3

Какие из приведенных ниже нотаций являются устоявшимися формами записи производной функции:

1. нотация Лейбница
2. нотация Лагранжа
3. нотация Гаусса
4. нотация Маркова

Правильные ответы: 1, 2

### 1.6.7 Резюме по разделу

В этом разделе мы поговорили о производных. Производная функции - это показатель скорости изменения значения функции в каждой ее точке. Математически производная функции  $f(x)$  в точке  $x_0$  определяется как предел отношения изменения значения функции  $f(x)$  к изменению ее аргумента  $x$  при стремлении изменения аргумента к нулю:

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

где  $\Delta x$  - бесконечно малое приращение аргумента (приращение, стремящееся к нулю)

Производные функций широко используются в машинном обучении. Например, производные могут использоваться для оптимизации функций потерь при обучении нейронных сетей. Оптимизация функций потерь требует нахождения экстремумов функции, что может быть сделано путем вычисления производных и использования методов оптимизации, таких как градиентный спуск. Производные также могут использоваться для нахождения экстремумов функций при обучении моделей машинного обучения, таких как линейная регрессия и логистическая регрессия.

## 1.7 Резюме по модулю

В этом модуле мы рассмотрели такие темы, как основные термины в области искусственного интеллекта и машинного обучения, библиотеки Python, используемые в рамках курса, математический аппарат, используемый в машинном обучении, векторы в и матрицы в линейной алгебре, функции и производные в математическом анализе.

Мы рассмотрели основные термины, которые широко используются в машинном обучении, включая искусственный интеллект, машинное обучение, глубинное обучение, наука о данных, анализ данных, обучение с учителем, обучение без учителя и обучение с подкреплением. Кроме того, мы ознакомились с основными специальными терминами, такими как объекты, признаки, целевая переменная, набор данных, модель, функция потерь, классификация, регрессия, кластеризация, снижение размерности.

Мы обсудили концепцию векторов. В математике вектор - это объект, который имеет определенную длину и направление в пространстве, и может быть представлен в виде упорядоченного набора чисел, называемых компонентами вектора. Свойства векторов включают длину, направление, операции суммы и разности, скалярное и векторное произведения, а также угол между векторами. В машинном обучении векторы используются для представления признаков или наблюдений в виде чисел. Например, вектор может представлять цвет изображения в RGB-формате, частоты звука или другие характеристики данных. Алгоритмы машинного обучения используют векторы для выполнения операций, таких как классификация, кластеризация и регрессия. Кроме того, векторы могут использоваться для решения задач оптимизации.

Мы изучили понятие матриц в контексте машинного обучения. Матрица - это таблица, состоящая из строк и столбцов, представляющая собой прямоугольный массив чисел. Некоторые основные свойства матриц:

- Размерность: количество строк и столбцов определяет размерность матрицы.
- Операции: матрицы можно складывать, вычитать, умножать на число и другие матрицы.
- Транспонирование: можно транспонировать матрицу, заменив строки на столбцы и наоборот.
- Обратная матрица: некоторые матрицы имеют обратную, которая используется для решения систем линейных уравнений.
- Определитель: число, вычисляемое для квадратной матрицы, может использоваться для решения систем линейных уравнений и для определения обратной матрицы.
- Ранг матрицы: максимальное число линейно независимых строк или столбцов в матрице.

Мы обсудили понятие функций. Функция - это правило, которое преобразует один набор значений (аргументы) в другой набор значений (значения функции). Функции могут быть определены математически, графически, в виде алгоритмов или программного кода. Некоторые из основных свойств функций:

- Гладкость функции: функция называется гладкой, если ее производные любого порядка непрерывны.
- Ограниченнность: свойство функции, которое описывает, насколько функция ограничена в своих значениях.
- Монотонность: свойство функции, которое описывает ее возрастание или убывание при изменении аргумента.

- Периодичность: свойство функции, которое описывает ее повторение с определенным периодом.
- Дифференцируемость: функция является дифференцируемой, если ее производная существует для всех точек в области определения.

Мы рассмотрели концепцию производных. Производная функции представляет собой индикатор скорости изменения значения функции в каждой ее точке при стремлении изменения аргумента к нулю:

$$f'(x) = \frac{f(x + \Delta x) - f(x)}{\Delta x}$$

где  $\Delta x$  - бесконечно малое приращение аргумента.

Производные функций находят широкое применение в области машинного обучения. Например, они используются для оптимизации функций потерь при обучении нейронных сетей.



# Глава 2

## Процесс разработки проекта машинного обучения. Подготовка данных. Оценка качества алгоритмов обучения с учителем. Метод ближайших соседей.

### 2.1 Процесс разработки проекта машинного обучения

Процесс разработки проекта машинного обучения обычно состоит из следующих шагов:

#### Этап 1: определение задачи

1. **Определение задачи:** на этом этапе вы определяете, какую задачу необходимо решить. Например, это может быть задача классификации или регрессии.

#### Этап 2: подготовка данных

1. **Сбор данных:** сбор данных, необходимых для обучения модели. Можно собрать данные самостоятельно или использовать уже существующие наборы данных.
2. **Исследование данных (англ. Exploratory Data Analysis, EDA):** это процесс исследования и анализа данных, используемый для получения полезной информации и выявления особенностей данных перед применением модели машинного обучения. EDA включает в себя методы визуализации и статистические техники, которые помогают понять данные и выделить важные характеристики.
3. **Очистка и исправление данных:** очистка данных от пропусков, дубликатов и ошибок.
4. **Разделение выборки:** разделение выборки на тренировочную и тестовую является важным шагом в машинном обучении, который помогает оценить качество модели на новых данных. Тренировочная выборка используется для обучения модели, тестовая выборка - для проверки ее качества.
5. **Предобработка признаков:** преобразование признаков, нормализация признаков и т.д.
6. **Создание новых признаков:** создание новых признаков - это важный шаг в построении моделей машинного обучения. Новые признаки могут улучшить качество модели и помочь ей лучше выявлять закономерности в данных. Существует несколько способов создания новых признаков:

- **Инженерия признаков на основе знаний об отрасли:** в этом подходе эксперты отрасли и/или аналитики создают новые признаки на основе своих знаний о предметной области. Это может включать в себя создание признаков на основе социально-экономических показателей, метеорологических данных, исторических событий и т.д.
- **Преобразование существующих признаков:** в этом подходе существующие признаки могут быть преобразованы в новые признаки путем использования математических операций, например, логарифмирование, возведение в степень, извлечение корня и т.д.
- **Создание признаков на основе текстов:** при анализе текстовых данных новые признаки могут быть созданы путем извлечения ключевых слов и фраз, использования стемминга, лемматизации, а также при помощи методов машинного обучения для анализа тональности и определения тем.
- **Создание признаков на основе временных рядов:** при анализе временных рядов новые признаки могут быть созданы путем вычисления статистических показателей, таких как скользящее среднее, стандартное отклонение, корреляция и т.д.

Важно понимать, что создание новых признаков может привести к увеличению размерности данных, что может усложнить обработку и анализ. Поэтому необходимо учитывать баланс между количеством признаков и качеством модели, а также проводить валидацию модели и контролировать переобучение.

7. **Отбор признаков:** отбор признаков может быть выполнен двумя способами: автоматически и вручную. Автоматический отбор признаков включает использование алгоритмов, которые оценивают важность каждого признака и выбирают только наиболее значимые. Вручную выбранные признаки обычно определяются на основе экспертного знания в предметной области.

### Этап 3: разработка модели МО

1. **Выбор модели:** на этом этапе вы выбираете модель машинного обучения, которая подходит для решения вашей задачи. В зависимости от задачи можно использовать различные типы моделей, такие как метрические модели, линейная регрессия, деревья решений и т.д.
2. **Обучение и настройка модели:** после выбора модели и подготовки данных вы можете обучить модель на тренировочных данных. Обычно это делается путем минимизации функции потерь с помощью градиентного спуска или других алгоритмов оптимизации. Если модель не дает достаточно хороших результатов, ее можно настроить, изменив гиперпараметры модели или используя другую модель.
3. **Оценка и валидация модели:** после обучения модели необходимо оценить ее качество на отложенных данных, которые модель не видела в процессе обучения. Это можно сделать с помощью различных метрик, таких как точность, F1-мера и т.д. После настройки модели ее необходимо проверить на тестовых данных, чтобы убедиться в ее работоспособности на новых данных.

### Этап 4: запуск модели

1. **Развертывание модели:** когда модель прошла все тесты и была удовлетворительно протестирована на новых данных, ее можно развернуть в производство, где она будет использоваться для решения реальных задач.

2. **Обслуживание модели:** после того, как модель развернута в производство, ее необходимо периодически обновлять и обслуживать, чтобы она продолжала давать актуальные результаты.

### 2.1.1 Вопросы для самопроверки

Какие задачи могут быть выбраны на этапе 1 (определение задачи) процесса разработки проекта машинного обучения:

1. классификация
2. регрессия
3. машинное обучение
4. численные методы

Правильные ответы: 1, 2

Этап 2 (подготовка данных) может включать в себя:

1. Сбор данных
2. Выбор модели
3. Разделение выборки
4. Создание новых признаков

Правильные ответы: 1, 3, 4

Выберите правильную последовательность шагов для этапа 3 (разработка модели МО):

1. Обучение и настройка модели -> Выбор модели -> Оценка и валидация модели
2. Выбор модели -> Обучение и настройка модели -> Оценка и валидация модели
3. Обучение и настройка модели -> Оценка и валидация модели -> Выбор модели
4. Оценка и валидация модели -> Обучение и настройка модели -> Выбор модели

Правильные ответы: 2

### 2.1.2 Резюме по разделу

Процесс разработки проекта машинного обучения состоит из нескольких этапов:

1. Определение задачи
2. Подготовка данных
3. Разработка модели МО
4. Запуск модели

В свою очередь эти этапы разбиваются на несколько шагов.

Этап 1: определение задачи

Этап 2: подготовка данных

1. Сбор данных
2. Исследование данных
3. Очистка и исправление данных

4. Разделение выборки
5. Предобработка признаков
6. Создание новых признаков
7. Отбор признаков

Этап 3: разработка модели МО

1. Выбор модели
2. Обучение модели
3. Оценка модели
4. Настройка модели
5. Валидация модели

Этап 4: запуск модели

1. Развёртывание модели
2. Обслуживание модели

## 2.2 Подготовка данных

Цель занятия: ученик может подготовить данные для моделей машинного обучения

### 2.2.1 Исследование данных

Исследование данных (EDA) позволяет понять данные, выявить скрытые связи и паттерны, определить выбросы и пропущенные значения, а также принять решение о необходимости дополнительной предобработки данных. EDA может включать в себя следующие шаги:

- **Визуализация данных.** Этот шаг включает в себя создание графиков и диаграмм, чтобы визуализировать данные и выявить скрытые закономерности и паттерны.
- **Изучение связей между различными переменными** и определение, какие переменные сильно влияют на другие.

В результате исследования данных мы получаем лучшее понимание данных, что помогает лучше выбрать подходящую модель машинного обучения, провести более эффективную предобработку данных и принять правильные решения на этапе обучения модели.

### 2.2.2 Очистка и исправление данных

Очистка данных - это процесс предварительной обработки данных, направленный на удаление или исправление ошибочных, неактуальных, неполных, поврежденных, дублированных или неправильно форматированных данных, чтобы улучшить качество и точность модели машинного обучения.

Ниже перечислены некоторые методы очистки данных в машинном обучении:

- **Удаление дубликатов.** Дубликаты могут привести к неверным результатам, поэтому необходимо удалить все дубликаты в данных.
- **Исправление ошибок.** В данных могут быть ошибки, например, опечатки или неверные значения. Такие ошибки могут быть исправлены с помощью автоматических методов, таких как исправление опечаток.

### 2.2.3 Разделение выборки на тренировочную и тестовую

Разбиение выборки на обучающую и тестовую является одним из ключевых этапов машинного обучения, который позволяет оценить качество работы модели на новых данных. При этом часть данных используется для обучения модели (обучающая выборка), а оставшаяся часть - для проверки ее качества (тестовая выборка). Благодаря такому разделению можно убедиться, что модель обладает высокой точностью на новых данных и не переобучена на имеющейся выборке. Важно помнить, что **разделение выборки на тренировочную и тестовую должно быть выполнено до начала обработки данных и обучения модели**. Также необходимо убедиться, что выборки не содержат дублирующихся или повторяющихся данных, и что разделение происходит случайным образом или с учетом определенного критерия, чтобы избежать смещения в оценке качества модели. Существует несколько способов разделения выборки на тренировочную, валидационную и тестовую в машинном обучении. Некоторые факторы, которые могут учитываться при выборе метода разбиения, включают в себя:

- **Размер выборки:** для небольших выборок может быть лучше использовать кросс-валидацию (перекрестную проверку), чтобы обеспечить более точную оценку модели.
- **Целевой признак:** если целевой признак несбалансированный, необходимо убедиться, что он представлен в обеих выборках.
- **Распределение признаков:** если признаки имеют разное распределение в тренировочной и тестовой выборках, это может привести к переобучению или недообучению модели.
- **Временные ряды:** при анализе временных рядов необходимо учитывать хронологический порядок данных и использовать методы, которые могут учитывать этот фактор.
- **Доступность данных:** иногда может быть сложно получить достаточное количество данных для построения отдельных тренировочной и тестовой выборок. В этом случае можно использовать методы, такие как кросс-валидация , чтобы максимизировать использование имеющихся данных.

В целом, выбор метода разбиения выборки зависит от конкретной задачи и доступных данных, и может потребовать экспериментов с различными методами, чтобы найти наилучший вариант.

Ниже перечислены наиболее распространенные из них:

- Разбиение на **обучающую, валидационную и тестовую** выборки в пропорции 60-20-20 (или любой другой соответствующий выбор) (функции train\_test\_split и test\_train\_split библиотеки sklearn)
- Использование **кросс-валидации**, при которой данные разбиваются на несколько равных частей (фолдов), и каждая часть используется в качестве тестовой выборки, а остальные части объединяются и используются для обучения модели (функции KFold, StratifiedKFold, TimeSeriesSplit и т.д. библиотеки sklearn)
- Однократное **разбиение на обучающую и тестовую выборки, а затем использование внутренней кросс-валидации** на обучающей выборке для подбора параметров модели и оценки ее качества на валидационной выборке.
- **Разбиение на обучающую, валидационную и тестовую выборки с помощью временных рядов**, когда данные разбиваются на обучающую и тестовую выборки в хронологическом порядке, а затем выборка для валидации формируется из части обучающей выборки, расположенной в более раннем временном периоде, чем тестовая выборка.

Кроме того, для некоторых задач может быть полезно использовать стратифицированное разбиение, когда сохраняется пропорция классов в каждой выборке, или разбиение с учетом баланса классов, когда классы распределяются между выборками с учетом их дисбаланса.

#### 2.2.4 Виды признаков

В машинном обучении существуют различные виды признаков, которые могут использоваться для описания объектов или данных. Некоторые из них:

- **Категориальные признаки:** признаки, которые принимают значения из определенного набора категорий или классов. Примерами категориальных признаков могут служить цвет, тип материала и т.д.
- **Числовые признаки:** признаки, которые принимают числовые значения, такие как длина, ширина, высота, возраст и т.д.

- **Бинарные признаки:** признаки, которые могут принимать только два значения, например, 0 и 1, да или нет, и т.д.
- **Текстовые признаки:** признаки, которые описывают текстовые данные, такие как заголовки новостей, описания продуктов и т.д.
- **Географические признаки:** признаки, которые описывают географические данные, такие как координаты, адреса и т.д.
- **Временные признаки:** признаки, которые описывают данные, относящиеся ко времени — дата, время, длительность и т.д.

Работа с разными видами признаков в машинном обучении может иметь свои особенности, так как каждый вид признаков имеет свои уникальные свойства и может требовать специфических методов работы.

- **Категориальные признаки:** Категориальные признаки нуждаются в преобразовании перед использованием в модели. Для этого могут использоваться различные методы, например, One-Hot Encoding или Label Encoding (определение этих методов будет рассмотрено ниже). Некоторые алгоритмы машинного обучения, такие как деревья решений и случайный лес, могут обрабатывать категориальные признаки напрямую.
- **Числовые признаки:** Числовые признаки могут быть использованы без какой-либо специальной обработки, но нормализация и стандартизация могут улучшить результаты. При работе с числовыми признаками также может быть полезно создать новые признаки, например, путем извлечения корня или возведения в квадрат.
- **Бинарные признаки:** Бинарные признаки могут быть использованы без специальной обработки, но могут потребовать обработки выбросов, так как значения могут быть ограничены только двумя значениями.
- **Текстовые признаки:** Текстовые признаки должны быть преобразованы в числовой формат для использования в модели. Для этого могут использоваться различные методы, такие как CountVectorizer или TF-IDF. CountVectorizer - это метод векторизации текста в машинном обучении, который используется для преобразования текстовых данных в векторы числовых значений, которые могут быть обработаны алгоритмами машинного обучения. CountVectorizer преобразует текстовые данные в векторы, где каждый элемент вектора соответствует количеству вхождений определенного слова в документе. Таким образом, каждый документ представлен в виде вектора, где размерность вектора равна общему числу уникальных слов во всех документах, а каждый элемент вектора соответствует количеству вхождений соответствующего слова в документ. TF-IDF (англ. Term Frequency-Inverse Document Frequency) - это метод векторизации текста, который используется для вычисления важности слова в документе на основе его частоты встречаемости в этом документе и общего количества документов, в которых это слово встречается. TF-IDF учитывает, что некоторые слова могут быть часто встречающимися во многих документах, и поэтому не могут быть использованы для отличия одного документа от другого. С другой стороны, редкие слова, которые встречаются только в некоторых документах, могут предоставить более ценную информацию для их классификации. TF-IDF вычисляется путем умножения «частоты слова в документе» (Term Frequency - TF) на «обратную частоту документа» (Inverse Document Frequency - IDF)
- **Географические признаки:** Географические признаки могут быть преобразованы в числовой формат, например, через координаты или почтовый индекс. При работе с

географическими признаками также могут быть полезны новые признаки, такие как расстояние между объектами или количество объектов в радиусе действия.

- **Временные признаки:** Временные признаки могут быть использованы без специальной обработки, но также могут быть полезны новые признаки, например, день недели или время суток.

## 2.2.5 Предобработка признаков

Предобработка признаков - это процесс подготовки данных перед применением модели машинного обучения. Цель предобработки признаков заключается в том, чтобы привести данные к такому виду, который будет оптимальным для обучения модели и получения максимальной точности предсказаний. Предобработка признаков является важным этапом в машинном обучении, так как позволяет улучшить точность модели, снизить шум и сделать данные более понятными для алгоритма. Некоторые этапы предобработки признаков в машинном обучении включают:

- Обработка выбросов, заполнение пропущенных значений и т.д.
- Преобразование признаков - преобразование данных в числовой формат, нормализация и масштабирование признаков, преобразование категориальных признаков и т.д.

### Обработка выбросов

Обработка выбросов в машинном обучении - это процесс идентификации и обработки экстремальных значений (наблюдений), которые могут исказить результаты анализа и привести к ошибочным выводам. Выбросы могут возникать из-за ошибок в измерениях, аномалий в данных или других факторов. Существует несколько способов обработки выбросов:

- **Замена выбросов:** подход заключается в замене выбросов на какое-то более типичное значение. Это может быть медиана, среднее значение или любое другое значение, которое соответствует типичным значениям признаков в выборке данных.
- **Удаление выбросов:** важно помнить, что удаление выбросов может привести к потере информации, поэтому решение о том, следует ли удалять выбросы, должно быть основано на конкретной задаче и анализе данных. Существует несколько способов удаления выбросов в машинном обучении, включая:
  - Метод межквартильного размаха (англ. Interquartile Range, IQR): Этот метод основан на вычислении межквартильного размаха данных, который определяет расстояние между 25-м и 75-м процентилем данных. Затем выбросы определяются как значения, находящиеся за пределами верхнего и нижнего порогов, определяемых как  $Q1 - 1.5 * IQR$  и  $Q3 + 1.5 * IQR$  соответственно.
  - Удаление выбросов на основе статистических критериев: Этот метод использует статистические критерии, такие как Z-оценка или Т-тест, для определения, является ли значение выбросом или нет. Если значение превышает определенный пороговый уровень, оно считается выбросом и удаляется.
- **Использование робастных (устойчивых к выбросам) моделей:** Робастные модели машинного обучения устойчивы к выбросам и могут работать более точно, даже если в выборке данных есть выбросы. Например, линейные модели, которые используют L1-регуляризацию (Lasso) или L2-регуляризацию (Ridge), могут быть более устойчивыми

к выбросам, чем стандартные линейные модели. Некоторые алгоритмы машинного обучения, такие как метод случайных деревьев и градиентный бустинг, могут учитывать выбросы, используя различные стратегии сэмплирования и ансамблирования.

- **Проведение дополнительного исследования:** Если выбросы вызваны реальными аномалиями, то они могут содержать важную информацию о данных. Поэтому может быть полезно провести дополнительное исследование, чтобы понять, почему выбросы возникли и как их можно использовать в моделировании.

### Заполнение пропущенных значений

Некоторые значения могут быть пропущены или недоступны, например, из-за ошибок или недоступности источника данных. Пропущенные значения можно заменить средним, медианным или модальным значением, или использовать другие методы заполнения, например, на основе предсказаний модели.

### Преобразование данных в числовой формат

Некоторые признаки могут иметь неправильный тип данных, например, строковые данные вместо числовых. Необходимо преобразовать типы данных в правильный формат. Для иллюстрации рассмотрим пример на основе признака «возраст» (Таблица 2.1):

Возраст	Возраст_-num
«20»	20
«30»	30
«40»	40
«50»	50

Таблица 2.1: Пример преобразования данных в числовой формат

### Нормализация и масштабирование признаков

Нормализация и масштабирование признаков - это процесс приведения значений признаков к определенному диапазону или масштабу. Этот процесс является важной частью предобработки данных в машинном обучении и может улучшить качество модели. Многие алгоритмы машинного обучения требуют, чтобы все признаки имели одинаковый масштаб, так как некоторые признаки могут иметь значительно большую вариативность, чем другие. Например, если у нас есть два признака: возраст в годах (от 0 до 100) и ежемесячный доход (от 0 до 1 000 000 рублей), то ежемесячный доход будет иметь значительно большую вариативность. Это может привести к тому, что алгоритмы машинного обучения будут отдавать большее значение признаку с большей вариативностью. Существуют несколько основных методов масштабирования признаков:

- **Нормализация (MinMax нормализация):** Этот метод преобразует значения признаков в диапазон от 0 до 1 или от -1 до 1. Это можно сделать с помощью формулы

2.1:

$$x_{norm} = \frac{x - x_{min}}{x_{max} - x_{min}}, \quad (2.1)$$

где  $x$  - это исходное значение,  $x_{min}$  и  $x_{max}$  - минимальное и максимальное значения в наборе данных соответственно, а  $x_{norm}$  - нормализованное значение.

Для иллюстрации рассмотрим пример на основе числового признака «возраст» (Таблица 2.2):

Возраст	MinMax нормали- зация
20	0.0000
30	0.3333
40	0.6667
50	1.0000

Таблица 2.2: Пример MinMax нормализации

Для каждого значения признака мы вычисляем новое значение, используя формулу, приведенную выше. В данном примере, минимальное значение возраста равно 20, а максимальное значение равно 50. При применении формулы, мы получаем новые значения признака, которые находятся в диапазоне от 0 до 1. Например, возраст 30 будет масштабирован до 0.3333, а возраст 50 будет масштабирован до 1.0000.

MinMax нормализация может быть полезна в случаях, когда значения признаков имеют разный масштаб и диапазон значений. Однако, необходимо быть осторожным при применении MinMax нормализации, если выборка содержит выбросы, так как они могут сильно искажать результаты масштабирования. Кроме того, необходимо обрабатывать случаи, когда в тестовой выборке встречаются значения признака, которых не было в обучающей выборке.

- **Стандартизация (L2 масштабирование, Z-оценка):** Этот метод преобразует значения признаков в распределение со средним значением 0 и стандартным отклонением 1. Это можно сделать с помощью формулы 2.2:

$$x_{std} = \frac{x - \mu}{\sigma}, \quad (2.2)$$

где  $x$  - значение признака,  $\mu$  - среднее значение признака в выборке,

$\sigma$  - стандартное отклонение признака в выборке.

Для иллюстрации рассмотрим пример на основе числового признака «возраст» (Таблица 2.3):

Для каждого значения признака мы вычисляем новое значение, используя формулу, приведенную выше.

В данном примере, среднее значение возраста равно 35, а стандартное отклонение равно 12.5. При применении формулы, мы получаем новые значения признака, которые находятся в стандартном нормальному распределении со средним значением равным 0 и стандартным отклонением равным 1. Например, возраст 30 будет масштабирован до

Возраст	Стандартизация
20	-1.3416
30	-0.4472
40	0.4472
50	1.3416

Таблица 2.3: Пример стандартизации

-0.4472, а возраст 50 будет масштабирован до 1.3416. Стандартизация может быть полезна в случаях, когда значения признаков имеют разный масштаб и диапазон значений, но при этом не содержат выбросы. Стандартизация также может улучшить качество модели, которая использует линейные методы машинного обучения, так как они часто основаны на предположении о стандартном нормальном распределении данных.

Выбор метода масштабирования признаков зависит от конкретной задачи и данных. Важно также отметить, что масштабирование признаков следует выполнять только на тренировочных данных, а затем применять те же параметры масштабирования на тестовых данных и новых данных в производственной среде.

### Преобразование категориальных признаков

Категориальные признаки (например, цвет, размер, марка автомобиля) являются одним из типов данных, используемых в машинном обучении. Однако, многие алгоритмы машинного обучения работают только с числовыми данными. Поэтому необходимо преобразовать категориальные признаки в числовые. Этот процесс называется кодированием категориальных признаков. Этот метод включает в себя преобразование категориальных признаков в числовые. Существует несколько методов кодирования категориальных признаков:

- **One-Hot Encoding:** Этот метод преобразует каждую категориальную переменную в набор бинарных переменных. Для каждого уникального значения переменной создается отдельная бинарная переменная. Если переменная принимает значение, равное значению категории, то соответствующая бинарная переменная будет равна 1, в противном случае - 0. Приведем пример One-Hot Encoding на основе категориального признака «фрукт» с тремя уникальными значениями: яблоко, банан и апельсин (Таблица 2.4).

Фрукт	Яблоко	Банан	Апельсин
Яблоко	1	0	0
Банан	0	1	0
Апельсин	0	0	1

Таблица 2.4: Пример One-Hot Encoding

Каждый уникальный фрукт превращается в отдельный столбец, который может быть 1 или 0 в зависимости от того, какой фрукт присутствует в конкретном наблюдении. Если в наблюдении присутствуют несколько фруктов, соответствующие столбцы получат значение 1. В приведенном выше примере первое и последнее наблюдение содержат яблоко, поэтому столбцы «Яблоко» для этих наблюдений имеют значение 1. Второе

и четвертое наблюдение содержат банан, поэтому соответствующий столбец «Банан» имеет значение 1. Третье и шестое наблюдение содержат апельсин, поэтому соответствующий столбец «Апельсин» имеет значение 1.

- **Label Encoding:** Этот метод присваивает каждому уникальному значению категориальной переменной уникальный целочисленный код. Приведем пример Label Encoding на основе категориального признака «фрукт» с тремя уникальными значениями: яблоко, банан и апельсин (Таблица 2.5). Каждая уникальная категория получает уникаль-

Фрукт	Код
Яблоко	1
Банан	2
Апельсин	3
Банан	2
Яблоко	1
Апельсин	3

Таблица 2.5: Пример Label Encoding

ный числовой код. В данном примере яблоко имеет код 1, банан имеет код 2, а апельсин имеет код 3. При использовании Label Encoding важно знать, что коды не являются порядковыми, то есть код 2 для банана не означает, что банан в два раза больше яблока. Коды используются только для идентификации категорий.

Label Encoding может быть полезен в случаях, когда категории обладают некоторой внутренней упорядоченностью, например, признак «образование», где можно использовать порядковые числа для идентификации уровня образования. Однако, если категории не обладают внутренней упорядоченностью, например, цвета, то использование Label Encoding может быть менее эффективным.

- **Mean Encoding (также Target Encoding):** Этот метод преобразования категориальных признаков в числовые значения, которые представляют среднее значение целевой переменной для каждой категории. Допустим, у нас есть набор данных с категориальным признаком «цвет» и целевой переменной «цена». Мы хотим закодировать категориальный признак «цвет» с помощью метода Mean Encoding.

1. Посчитаем среднее значение целевой переменной для каждой категории «цвета» (Таблица 2.6):

Цвет	Цена
Красный	10
Зеленый	20
Синий	30
Красный	15
Зеленый	25

Таблица 2.6: Начальные значения для каждой категории

Средние значения для каждой категории (Таблица 2.7):

2. Заменим значения категориального признака «цвет» на его средние значения (Таблица 2.8):

Цвет	Цена
Красный	12.5
Зеленый	22.5
Синий	30.0

Таблица 2.7: Средние значения для каждой категории

Цвет	Цена
Красный	12.5
Зеленый	22.5
Синий	30.0
Красный	12.5
Зеленый	22.5

Таблица 2.8: Закодированные значения для каждой категории

Таким образом, мы закодировали категориальный признак «цвет» в числовые значения с помощью метода Mean Encoding, используя информацию о целевой переменной «цена». Это может помочь улучшить точность модели машинного обучения при работе с категориальными признаками.

- **Frequency Encoding:** Этот метод присваивает каждому уникальному значению категориальной переменной частоту его появления в данных. Вот пример Frequency Encoding на основе категориального признака «фрукт» с тремя уникальными значениями: яблоко, банан и апельсин (Таблица 2.9):

Фрукт	Frequency Encoding
Яблоко	0.3333
Банан	0.3333
Апельсин	0.3333
Банан	0.3333
Яблоко	0.3333
Апельсин	0.3333

Таблица 2.9: Пример Label Encoding

Каждое уникальное значение категориального признака заменяется на частоту его появления в данных. В данном примере все три значения фруктов встречаются по два раза, поэтому их частота равна  $2/6 = 0.3333$ . В строках, где фрукт повторяется, значения Frequency Encoding остаются одинаковыми.

Frequency Encoding может быть полезен в случаях, когда категории не имеют внутренней упорядоченности, а частота появления категории в данных может быть связана с целевой переменной. Однако, при использовании Frequency Encoding необходимо быть осторожным, чтобы избежать переобучения модели. Кроме того, необходимо обработать случаи, когда в тестовой выборке встречаются значения категориального признака, которых не было в обучающей выборке.

Эти методы могут быть применены как отдельно, так и в сочетании друг с другом, в зависимости от исходных данных. Преобразование признаков является важным шагом в процессе подготовки данных для модели машинного обучения и может существенно улучшить качество модели.

## 2.2.6 Создание новых признаков

Создание новых признаков (англ. Feature Engineering) в машинном обучении - это процесс преобразования исходных данных в новые признаки, которые могут улучшить качество модели и улучшить ее способность к предсказанию. Ниже перечислены некоторые методы создания новых признаков в машинном обучении:

- **Инженерия признаков на основе знаний об отрасли:** в этом подходе эксперты отрасли и/или аналитики создают новые признаки на основе своих знаний о предметной области. Это может включать в себя создание признаков на основе социально-экономических показателей, метеорологических данных, исторических событий и т.д. Например, в медицинских данных может быть создан признак «Индекс массы тела» (BMI), используя формулу на основе роста и веса пациента.
- **Преобразование существующих признаков:** в этом подходе существующие признаки могут быть преобразованы в новые признаки путем использования математических операций, например, логарифмирование, возведение в степень, извлечение корня и т.д. Можно создавать новые признаки, например, путем извлечения значений из других признаков. Например, можно извлечь день недели или время из даты и времени.
- **Создание признаков на основе текстов:** при анализе текстовых данных новые признаки могут быть созданы путем извлечения ключевых слов и фраз, использования стемминга, лемматизации, CountVectorizer, TF-IDF, BPE, а также при помощи методов машинного обучения для анализа тональности и определения тем.
- **Создание признаков на основе временных рядов:** при анализе временных рядов новые признаки могут быть созданы путем вычисления статистических показателей, таких как скользящее среднее, стандартное отклонение, корреляция и т.д.

## 2.2.7 Отбор признаков

Отбор признаков (англ. Feature Selection) в машинном обучении - это процесс выбора наиболее значимых признаков из множества доступных, которые будут использоваться для обучения модели. Цель отбора признаков состоит в том, чтобы уменьшить размерность данных, устранив шум, повысить точность модели и улучшить ее интерпретируемость. Существует несколько методов отбора признаков:

- **Рекурсивное устранение признаков (англ. Recursive Feature Elimination):** используется алгоритм, который удаляет признаки с наименьшей важностью для модели, пока не останется необходимое число признаков.
- **Подход на основе важности признаков:** определяется важность признаков для модели с помощью алгоритмов, таких как случайный лес или градиентный бустинг. На основе этого отбираются наиболее важные признаки.
- **Подход на основе корреляции признаков:** удаляются признаки, которые сильно коррелируют друг с другом, так как они могут увеличивать сложность модели без улучшения ее качества.

Важно понимать, что отбор признаков - это процесс, который может повлиять на точность модели. Поэтому необходимо тщательно оценить каждый метод и выбрать наиболее подходящий в каждой конкретной ситуации.

### 2.2.8 Реализация в Python

Реализация основных этапов подготовки данных для проекта машинного обучения рассмотрена в jupyter notebook файле: lecture\_02\_code\_labs\_01\_pandas.ipynb.

### 2.2.9 Вопросы для самопроверки

Исследование данных в процессе разработки проекта машинного обучения может включать в себя:

1. визуализацию данных
2. заполнение пропущенных значений
3. создание новых признаков
4. изучение связей между различными переменными

Правильные ответы: 1, 4

Выберите, какие из приведенных ниже признаков являются категориальными:

1. Пол: мужской/женский
2. Цвет: красный/зеленый/синий/желтый
3. Марка автомобиля: BMW/Mercedes/Audi
4. Образование: высшее/среднее/начальное
5. Страна происхождения: Россия/США/Китай
6. Продукты питания: молоко/сыр/мясо/овощи/фрукты
7. Уровень дохода: низкий/средний/высокий
8. Домашние животные: кошка/собака/хомяк

Правильные ответы: 2, 3, 5, 6, 8 Пояснение: 1 вариант - бинарный признак, 4 и 7 варианты обладают внутренней упорядоченностью

Какой из приведенных ниже методов преобразует каждую категориальную переменную в набор бинарных переменных:

1. Frequency Encoding
2. Label Encoding
3. One-Hot Encoding
4. Mean Encoding

Правильные ответы: 3

### 2.2.10 Резюме по разделу

На текущий момент мы рассмотрели основные этапы подготовки данных для проекта машинного обучения, а именно:

**Исследование данных:** различные методы визуализации и статистические техники

**Очистка данных:** очистка данных от ошибок и пропусков

**Разделение выборки:** Разбиение выборки на обучающую и тестовую является одним из важных этапов машинного обучения. Важно помнить, что разделение выборки на тренировочную и тестовую должно быть выполнено до начала обработки данных и обучения модели.

**Работа с признаками:** работа с выбросами и преобразование признаков

**Создание новых признаков:** процесс выбора наиболее значимых признаков из множества доступных, которые будут использоваться для обучения модели

**Отбор признаков:** процесс выбора наиболее значимых признаков из множества доступных, которые будут использоваться для обучения модели.

## 2.3 Оценка качества алгоритмов обучения с учителем

**Цель занятия:** ученик может самостоятельно подобрать подходящую слушаю метрику качества.

План занятия:

1. Метрики качества регрессии
2. Метрики качества классификации
3. Обобщающая способность алгоритмов

Прежде чем мы начнем работать с алгоритмами машинного обучения, давайте поймем, как можно оценивать качество работы алгоритмов машинного обучения. Это позволит в дальнейшем оценивать качество всех алгоритмов, с которыми мы будем иметь дело. Для этой цели используются специальные формулы, называемые метриками качества.

Метрики качества в машинном обучении нужны для оценки качества работы алгоритмов. Они позволяют сравнивать разные модели машинного обучения и выбирать наилучшую для конкретной задачи. Кроме того, метрики качества позволяют определить, насколько хорошо обученная модель справляется с поставленной перед ней задачей и насколько ее результаты достоверны.

Например, метрики качества для задачи классификации позволяют определить, насколько точно модель предсказывает метки классов для тестовых данных. А метрики качества для задачи регрессии позволяют определить, насколько близко предсказанные значения к истинным.

Метрики качества могут быть использованы для выбора оптимального набора гиперпараметров<sup>1</sup> модели, для настройки ее параметров, а также для сравнения разных алгоритмов машинного обучения и выбора наилучшего из них.

### 2.3.1 Метрики качества регрессии

Метрики качества регрессии - это числовые характеристики, которые используются для оценки того, насколько хорошо модель регрессии соответствует данным. Ниже приведены некоторые из наиболее распространенных метрик качества регрессии с примерами:

- **Среднеквадратичная ошибка (англ. Mean Square Error, MSE)** - это среднее значение квадратов разностей между прогнозируемыми значениями и фактическими значениями.

Формула для средней квадратичной ошибки (MSE) выглядит следующим образом (Формула 2.3):

$$MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2, \quad (2.3)$$

где  $y_i$  - это наблюдаемое значение, а  $\hat{y}_i$  - это предсказанное значение.

MSE особенно часто используется, когда существует линейная зависимость между признаками и целевой переменной. Это происходит, когда мы имеем дело с непрерывными переменными, такими как цена на недвижимость или доход.

---

<sup>1</sup>Гиперпараметры модели - это параметры, которые определяются еще до обучения модели. Они определяют поведение алгоритма обучения, такие как скорость обучения, количество скрытых слоев в нейронной сети, количество деревьев в случайном лесу и т.д.

Пример: Если у нас есть модель, которая предсказывает количество продаж в магазине, MSE может быть рассчитана как среднее значение квадратов разностей между прогнозируемыми продажами и фактическими продажами.

- **Средняя абсолютная ошибка (англ. Mean Absolute Error, MAE)** - это среднее значение абсолютных разностей между прогнозируемыми значениями и фактическими значениями. Формула для средней абсолютной ошибки (MAE) выглядит следующим образом (Формула 2.4):

$$MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{y}_i|, \quad (2.4)$$

где  $y_i$  - это наблюдаемое значение, а  $\hat{y}_i$  - это предсказанное значение.

MAE чаще используется, когда мы имеем дело с выбросами и выбивающимися значениями. Она более устойчива к выбросам, чем MSE.

Пример: Предположим, что у нас есть модель, которая прогнозирует цены на недвижимость. MAE для этой модели может быть рассчитана как среднее значение абсолютных разностей между прогнозируемыми ценами и реальными ценами.

- **Функция потерь Хьюбера (англ. Huber loss)** - это функция потерь, которая комбинирует свойства среднеквадратичной ошибки (MSE) и средней абсолютной ошибки (MAE) в зависимости от значения отклонения. Она используется для задач регрессии и оптимизации моделей машинного обучения. Формула для функции потерь Хьюбера выглядит следующим образом (Формула 2.5):

$$L_\delta(y_i, \hat{y}_i) = \begin{cases} \frac{1}{2}(y_i - \hat{y}_i)^2 & \text{если } |y_i - \hat{y}_i| \leq \delta \\ \delta(|y_i - \hat{y}_i| - \frac{1}{2}\delta) & \text{иначе} \end{cases}$$

где  $y_i$  - это наблюдаемое значение, а  $\hat{y}_i$  - это предсказанное значение,

$\delta$  - параметр для определения порога между квадратичной и линейной потерями.

(2.5)

Функция потерь Хьюбера вычисляется как среднее значение потерь для всех примеров. Функция потерь Хьюбера более устойчива к выбросам, чем MSE, потому что она уменьшает вклад выбросов в общую ошибку. При этом, она сохраняет свойства MSE, когда отклонение мало, и свойства MAE, когда отклонение большое.

Пример: Предположим, что у нас есть модель, которая прогнозирует цены на недвижимость. Huber loss для этой модели может быть рассчитана как среднее значение функции потерь Хьюбера между прогнозируемыми ценами и реальными ценами. Параметр  $\delta$  можно выбрать в зависимости от того, насколько сильно мы хотим уменьшить влияние выбросов на общую ошибку.

Это только некоторые из наиболее распространенных метрик качества регрессии. В зависимости от задачи и данных, могут быть использованы и другие метрики.

## Отображение метрик качества регрессии

В основе MSE, MAE и функции потерь Хьюбера лежит вычислительный процесс, который учитывает отклонения наблюдений от аппроксимирующей гиперплоскости (Рисунок 2.1).

Отображение графика отклонений от аппроксимирующей линии для MSE и MAE позволяет оценить точность модели и выявить наличие систематических ошибок в предсказаниях.

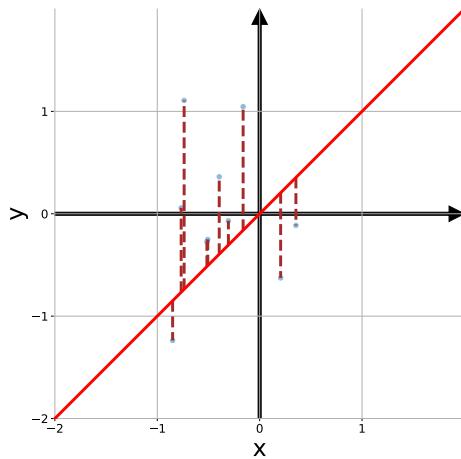


Рисунок 2.1: Визуализация вычисления метрик качества регрессии

Если на графике присутствует какая-то структура или зависимость между отклонениями и значением переменных, это может указывать на недостатки модели и необходимость ее улучшения. Если же график показывает случайное распределение отклонений вокруг нуля, то это свидетельствует о том, что модель хорошо справляется с предсказаниями.

### 2.3.2 Метрики качества классификации

Метрики качества классификации - это числовые характеристики, которые используются для оценки того, насколько хорошо модель классификации соответствует данным.

#### Концепция TP, TN, FP, FN в метриках качества классификации

Для работы с метриками качества классификации необходимо понимать такие понятия, как **истинно-положительное решение** (англ. True Positive, TP), **истинно-отрицательное решение** (англ. True Negative, TN), **ложно-положительное решение** (англ. False Positive, FP) и **ложно-отрицательное решение** (англ. False Negative, FN).

TP, TN, FP, FN - это часто используемые сокращения для обозначения результатов двоичной классификации, где у нас есть два класса: положительный (P) и отрицательный (N).

**TP** - это количество правильно классифицированных положительных примеров (True Positive). Например, если модель должна классифицировать изображения на котов и собак, и она правильно классифицирует изображение кота как кота, это будет являться TP.

**TN** - это количество правильно классифицированных отрицательных примеров (True Negative). Например, если модель должна классифицировать изображения на котов и собак, и она правильно классифицирует изображение автомобиля как не являющееся ни котом, ни собакой, это будет являться TN.

**FP** - это количество неправильно классифицированных положительных примеров (False Positive). Например, если модель должна классифицировать изображения на котов и собак, и она неправильно классифицирует изображение собаки как кота, это будет являться FP.

**FN** - это количество неправильно классифицированных отрицательных примеров (False Negative). Например, если модель должна классифицировать изображения на котов и собак, и она неправильно классифицирует изображение кота как не являющееся ни котом, ни собакой, это будет являться FN.

Рассмотрим наглядный пример. Допустим, у нас есть модель для определения, является ли письмо не спамом (хорошее письмо) или спамом.

- True Positive (TP) - это случай, когда письмо является хорошим и наша модель правильно предсказывает его как хорошее.
- True Negative (TN) - это случай, когда письмо является спамом и наша модель правильно предсказывает его как спам.
- False Positive (FP) - это случай, когда письмо является спамом, но наша модель ошибочно предсказывает его как хорошее.
- False Negative (FN) - это случай, когда письмо является хорошим, но наша модель ошибочно предсказывает его как спам.

Эти примеры можно обобщить с помощью матрицы ошибок (Таблица 2.10):

		Наблюдаемое значение	
		Хорошее	Спам
Предсказанное значение	Хорошее	TP	FP
	Спам	FN	TN

Таблица 2.10: Матрица ошибок (Confusion matrix)

Давайте рассмотрим несколько примеров:

- Пример 1: у нас есть письмо, которое является хорошим, и наша модель правильно предсказывает его как хорошее. В этом случае, это будет True Positive (TP).
- Пример 2: у нас есть письмо, которое является спамом, но наша модель ошибочно предсказывает его как хорошее. В этом случае, это будет False Positive (FP) (ошибка I рода, ложное срабатывание).
- Пример 3: у нас есть письмо, которое является спамом, и наша модель правильно предсказывает его как спам. В этом случае, это будет True Negative (TN).
- Пример 4: у нас есть письмо, которое является хорошим, но наша модель ошибочно предсказывает его как спам. В этом случае, это будет False Negative (FN) (ошибка II рода).

В машинном обучении и статистике используется несколько различных метрик для измерения точности моделей. Рассмотрим основные из них.

### Accuracy для бинарной классификации

**Accuracy (доля правильных ответов)** - отношение числа правильно классифицированных объектов к общему числу объектов в выборке в задаче бинарной классификации. Accuracy - это самая простая метрика, которая вычисляет долю верно классифицированных

объектов в общем числе объектов. Она часто используется для оценки качества работы алгоритмов классификации сбалансированных классов. Формула для accuracy (Формула 2.6):

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN},$$

где  $TP$  - количество верно предсказанных положительных классов,

$TN$  - количество верно предсказанных отрицательных классов,

$FP$  - количество ложно предсказанных положительных классов,

$FN$  - количество ложно предсказанных отрицательных классов.

Рассмотрим два примера задачи бинарной классификации для выявления спама. Первый пример - описательный, второй - с визуализацией. Пусть у нас есть набор данных, состоящий из 1000 писем, из которых 950 - хорошие письма, а 50 - письма-спам. В этом случае, классы несбалансированы, потому что спама гораздо меньше, чем хороших писем. Если мы используем accuracy как метрику для оценки качества модели, то в этом случае accuracy может давать искаженные результаты. Например, допустим, что модель предсказывает, что все письма - хорошие, тогда accuracy составит 0.95, что может показаться высокой точностью модели. Однако, такая модель совершенно не пригодна для решения поставленной задачи, так как ее цель - выявление спама и она не сможет правильно классифицировать такие объекты.

Рассмотрим второй пример, где писем поровну - по 10. Визуализация вычисления метрики accuracy для такого примера представлена на рисунке 2.2.

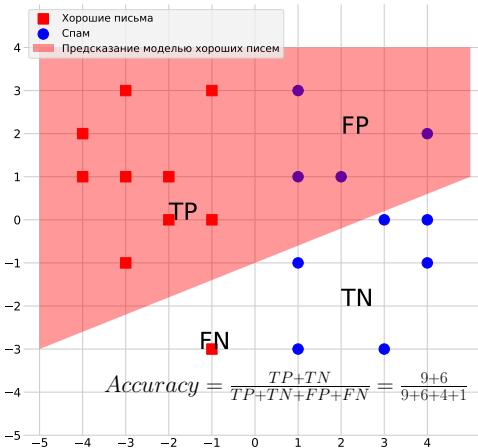


Рисунок 2.2: Визуализация вычисления метрики accuracy для бинарной классификации

### Precision для бинарной классификации

**Precision (точность)** - отношение числа верно предсказанных положительных классов к общему числу положительных предсказаний в задаче бинарной классификации. Метрика точности (precision) используются для оценки качества моделей машинного обучения в задачах классификации. Они позволяют измерять, насколько хорошо модель выделяет объекты определенного класса из общего числа объектов, которые она относит к этому классу.

Формула precision (Формула 2.7):

$$Precision = \frac{TP}{TP + FP}, \quad (2.7)$$

где  $TP$  - количество верно предсказанных положительных классов,

$FP$  - количество ложно предсказанных положительных классов.

Продолжая пример с задачей выявления спама, рассмотрим метрику точности (precision) для этой задачи. В данном случае, точность определяет, какой процент из всех писем, которые модель предсказала как спам, действительно являются спамом. Допустим, модель классифицирует 70 писем как спам, и из них на самом деле только 50 писем являются мошенническими. Тогда точность будет равна  $50/70 = 0.71$ , что означает, что модель правильно классифицировала 71% писем, которые она отнесла к классу спам. Метрика точности позволяет оценить качество работы модели с точки зрения ее способности правильно идентифицировать объекты положительного класса, т.е. спам в нашем примере. Однако, метрика точности может не учитывать количество ложно-отрицательных прогнозов, т.е. случаев, когда модель не определила письма как мошеннические, но на самом деле они являются мошенническими. Для полной оценки качества работы модели необходимо рассматривать и другие метрики, такие как recall (полнота), которая показывает, какой процент из всех мошеннических писем модель способна правильно идентифицировать, и F1-score, который является гармоническим средним между precision и recall и учитывает как точность, так и полноту модели.

Визуализация вычисления метрики precision для набора из 20 писем представлена на рисунке 2.3.

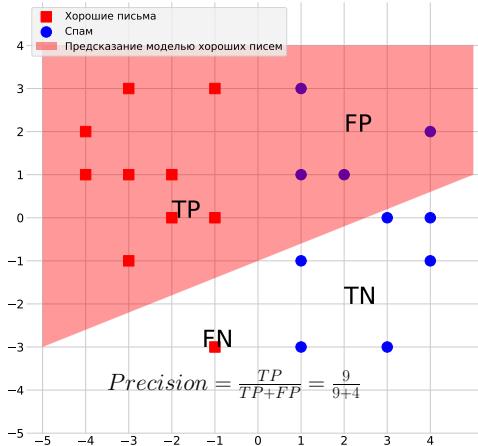


Рисунок 2.3: Визуализация вычисления метрики precision для бинарной классификации

### Recall (полнота) для бинарной классификации

**Recall (полнота)** - отношение числа верно предсказанных положительных классов к общему числу реальных положительных классов. Метрика полноты (recall) также используется для оценки качества моделей машинного обучения в задачах классификации. Она позволяет измерять, насколько хорошо модель распознает объекты определенного класса из общего

числа объектов, которые принадлежат этому классу (Формула 2.8):

$$Recall = \frac{TP}{TP + FN}, \quad (2.8)$$

где  $TP$  - количество верно предсказанных положительных классов,

$FN$  - количество ложно предсказанных положительных классов.

Продолжая пример с задачей выявления спама, рассмотрим метрику полноты (recall) для этой задачи. В данном случае, полнота определяет, какой процент из всех мошеннических писем модель способна правильно идентифицировать. Допустим, что в общем числе 100 писем модель смогла правильно идентифицировать только 50. Тогда полнота будет равна  $50/100 = 0.5$ , что означает, что модель правильно распознала только 50% всех мошеннических писем. Метрика полноты позволяет оценить качество работы модели с точки зрения ее способности правильно идентифицировать все объекты положительного класса, т.е. все письма-спам в нашем примере. Однако, метрика полноты может не учитывать количество ложно-положительных прогнозов, т.е. случаев, когда модель неправильно идентифицирует операции как мошеннические, когда на самом деле они ими не являются.

Визуализация вычисления метрики recall для примера с 20 письмами представлена на рисунке 2.4.

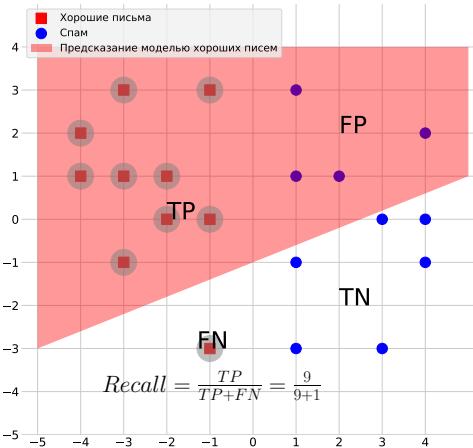


Рисунок 2.4: Визуализация вычисления метрики recall для бинарной классификации

### F1-мера для бинарной классификации

F1 score - это мера точности и полноты модели, которая вычисляется как гармоническое среднее точности (precision) и полноты (recall) модели. Формула для расчета F1 score в задаче бинарной классификации (Формула 2.9):

$$F1 = 2 * \frac{precision * recall}{precision + recall} = \frac{TP}{TP + \frac{1}{2}(FP + FN)},$$

где  $TP$  - количество верно предсказанных экземпляров положительного класса,  $(2.9)$

$FP$  - количество ложно предсказанных экземпляров положительного класса,

$FN$  - количество ложно предсказанных экземпляров отрицательного класса.

В бинарной классификации 1000 писем для каждого письма алгоритм определяет, является ли оно спамом или нет. Давайте зададим другое распределение спама и хороших писем, 500 являются спамом, а 500 - нет. Допустим, алгоритм классификации верно определил 450 спам-писем и 480 хороших писем.  $TP = 450$ ,  $TN = 480$ ,  $FP = 20$ ,  $FN = 50$ . Тогда:  $Accuracy = (TP + TN) / (TP + TN + FP + FN) = (450 + 480) / (450 + 480 + 20 + 50) = 0.93$   $Precision = TP / (TP + FP) = 450 / (450 + 20) = 0.96$   $Recall = TP / (TP + FN) = 450 / (450 + 50) = 0.9$   $F1\text{-score} = 2 * Precision * Recall / (Precision + Recall) = 2 * 0.96 * 0.9 / (0.96 + 0.9) = 0.93$

Алгоритм показывает высокую точность и полноту, так как значение accuracy, precision, recall и F1-score достаточно высокие (больше 0.9). Это говорит о том, что алгоритм хорошо находит и отличает мошеннические письма от обычных, и его можно использовать для дальнейшей работы с данными. Однако, возможно, стоит обратить внимание на то, что FN (ложноотрицательные результаты) выше, чем FP (ложноположительные результаты), что может быть проблемой, если ложноотрицательные результаты критичны для данной задачи. В целом, для задачи бинарной классификации определения спама результаты работы алгоритма можно считать хорошими.

Визуализация вычисления метрики F1 для примера с 20 письмами представлена на рисунке 2.5.

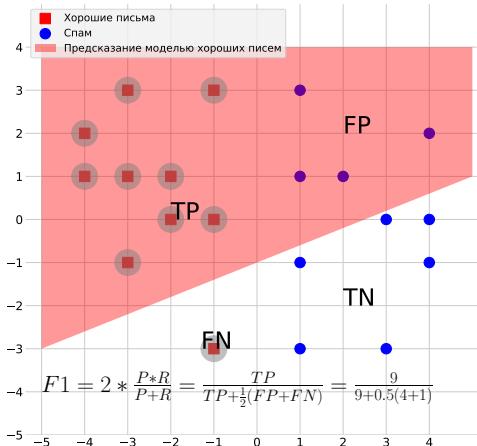


Рисунок 2.5: Визуализация вычисления метрики F1 для бинарной классификации

## Многоклассовая классификация

В целом, принцип выбора метрик в задаче многоклассовой классификации совпадает с задачей бинарной классификации. Однако, в задаче многоклассовой классификации есть несколько модификаций метрик precision, recall и F1: micro, macro и weighted. Главным отличием между micro, macro и weighted модификациями является то, как они учитывают распределение классов. Модификация micro учитывает все истинно-положительные и ложно-положительные примеры, модификация macro не учитывает несбалансированность классов в наборе данных, модификация weighted учитывает этот фактор.

Стоит отметить, что если классы сбалансиированы, в задаче многоклассовой классификации различные модификации метрик (micro, macro и weighted усреднение) будут показывать практически идентичный результат.

Для объяснения вычисления метрик многоклассовой классификации возьмем следующий пример (2.6).

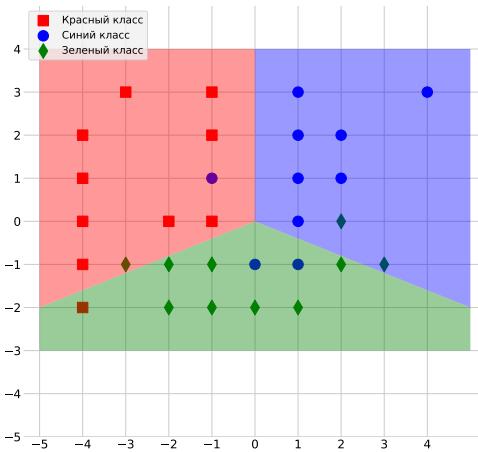


Рисунок 2.6: Матрица ошибок

Будем считать ответы, двигаясь по сетке слева-направо сверху-вниз Получим такие наборы данных (1 - красный квадрат, 2 - синий круг, 3 - зеленый ромб):

Истинные значения = [1, 1, 1, 1, 1, 1, 3, 1, 3, 3, 1, 1, 2, 1, 3, 3, 2, 3, 2, 2, 2, 2, 3, 2, 2, 3, 3, 3, 2]

Предсказанные значения = [1, 1, 1, 1, 3, 1, 1, 1, 3, 3, 1, 1, 1, 1, 3, 3, 3, 2, 2, 2, 2, 3, 3, 2, 2, 3, 2, 2]

Создадим матрицу ошибок для вычисления метрик многоклассовой классификации (2.7).

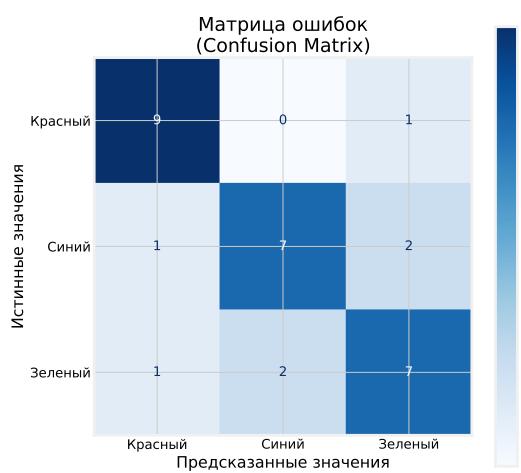


Рисунок 2.7: Матрица ошибок

## Accuracy для многоклассовой классификации

Формулу для accuracy (Формула 2.6) можно использовать и для многоклассовой классификации. Мы просто считаем количество правильно и неправильно классифицированных объектов. По матрице ошибок можно получить значение accuracy (сумма значений по диагоналям / сумма значений по матрице):

$$\text{accuracy} = \frac{9 + 7 + 7}{9 + 7 + 7 + 0 + 1 + 2 + 1 + 1 + 2} = 0.77$$

Стоит заметить, что accuracy плохо работает на несбалансированных выборках.

## Precision для многоклассовой классификации

Для одного класса  $c$  в задаче многоклассовой классификации precision вычисляется по формуле 2.10:

$$\text{precision}_c = \frac{TP_c}{TP_c + FP_c}, \quad (2.10)$$

где  $TP_c$  - количество верно предсказанных положительных примеров класса  $c$ ,

$FP_c$  - количество ложно предсказанных положительных примеров класса  $c$ .

**Micro precision** - это метрика, которая вычисляет точность для каждого класса по отдельности, а затем усредняет их с использованием общего числа истинных и ложных положительных и отрицательных ответов по всем классам. Эта метрика подходит для задач, где классы не сбалансираны и модель должна давать одинаковый вес каждому объекту. Формула для вычисления micro precision (Формула 2.11):

$$\text{precision}_{\text{micro}} = \frac{\sum_c TP_c}{\sum_c (TP_c + FP_c)}, \quad (2.11)$$

где  $TP_c$  - количество верно предсказанных положительных примеров класса  $c$ ,

$FP_c$  - количество ложно предсказанных положительных примеров класса  $c$ .

В нашем примере вычислить micro precision можно с помощью матрицы ошибок:

$$\text{precision}_{\text{micro}} = \frac{9 + 7 + 7}{(9 + 0 + 1) + (7 + 1 + 2) + (7 + 1 + 2)} = 23/30 = 0.766$$

**Macro precision** - это метрика, которая вычисляет точность для каждого класса по отдельности и затем усредняет их. Эта метрика не учитывает размеры классов, поэтому она подходит для задач, где классы сбалансираны. Формула для вычисления macro precision (Формула 2.12):

$$\text{precision}_{\text{macro}} = \frac{1}{C} \sum_c P_c, \quad (2.12)$$

где  $P_c$  - точность по классу  $c$ ,  $C$  - количество классов в задаче классификации.

В нашем примере вычислить macro precision можно с помощью матрицы ошибок:

$$\text{precision}_{\text{macro}} = \frac{\frac{9}{9+1+1} + \frac{7}{7+0+2} + \frac{7}{7+1+2}}{3} = 0.765$$

**Weighted precision** - это метрика, которая вычисляет точность для каждого класса по отдельности и затем усредняет их с использованием весов, пропорциональных размеру каждого класса. Эта метрика подходит для задач, где классы не сбалансированы, и модель должна учитывать важность каждого класса. Формула для вычисления weighted precision (Формула 2.13):

$$precision_{weighted} = \frac{\sum_c (TP_c + FP_c) \cdot P_c}{\sum_c (TP_c + FP_c)},$$

где  $P_c$  - точность по классу  $c$ , (2.13)

$TP_c$  - количество верно предсказанных положительных примеров класса  $c$ ,

$FP_c$  - количество ложно предсказанных положительных примеров класса  $c$ .

В нашем примере вычислить weighted precision можно с помощью матрицы ошибок:

$$precision_{weighted} = \frac{(9 + 1 + 1) * \frac{9}{9+1+1} + (7 + 0 + 2) * \frac{7}{7+0+2} + (7 + 1 + 2) * \frac{7}{7+1+2}}{(9 + 1 + 1) + (7 + 0 + 2) + (7 + 1 + 2)} = 0.765$$

### Recall (полнота) для многоклассовой классификации

Для одного класса  $c$  в задаче многоклассовой классификации **Recall** вычисляется по формуле 2.14:

$$recall_c = \frac{TP_c}{TP_c + FN_c},$$

где  $TP_c$  - количество верно предсказанных положительных примеров класса  $c$ ,

$FP_c$  - количество ложно предсказанных положительных примеров класса  $c$ .

**Micro recall** - это метрика, которая вычисляет полноту для каждого класса по отдельности, а затем усредняет их с использованием общего числа истинных и ложных положительных и отрицательных ответов по всем классам. Эта метрика подходит для задач, где классы не сбалансированы и модель должна давать одинаковый вес каждому объекту. Формула для вычисления micro recall (Формула 2.15):

$$recall_{micro} = \frac{\sum_c TP_c}{\sum_c (TP_c + FN_c)},$$

где  $TP_c$  - количество верно предсказанных положительных примеров класса  $c$ ,

$FN_c$  - количество ложно предсказанных отрицательных примеров класса  $c$ .

В нашем примере вычислить micro recall можно с помощью матрицы ошибок:

$$recall_{micro} = \frac{9 + 7 + 7}{(9 + 0 + 1) + (7 + 1 + 2) + (7 + 1 + 2)} = 23/30 = 0.766$$

**Macro recall** - это метрика, которая вычисляет полноту для каждого класса по отдельности и затем усредняет их. Эта метрика не учитывает размеры классов, поэтому она подходит для задач, где классы сбалансированы. Формула для вычисления macro recall (Формула 2.16):

$$recall_{macro} = \frac{1}{C} \sum_c R_c,$$

где  $R_c$  - полнота по классу  $c$ ,  $C$  - количество классов в задаче классификации.

В нашем примере вычислить macro recall можно с помощью матрицы ошибок:

$$\text{recall}_{\text{micro}} = \frac{\frac{9}{9+0+1} + \frac{7}{7+1+2} + \frac{7}{7+1+2}}{3} = 23/30 = 0.766$$

**Weighted recall** - это метрика, которая вычисляет полноту для каждого класса по отдельности и затем усредняет их с использованием весов, пропорциональных размеру каждого класса. Эта метрика подходит для задач, где классы не сбалансированы, и модель должна учитывать важность каждого класса. Формула для вычисления weighted recall (Формула 2.17):

$$\text{recall}_{\text{weighted}} = \frac{\sum_c (TP_c + FN_c) \cdot R_c}{\sum_c (TP_c + FN_c)},$$

где  $R_c$  - полнота по классу  $c$ , (2.17)

$TP_c$  - количество верно предсказанных положительных примеров класса  $c$ ,

$FN_c$  - количество ложно предсказанных отрицательных примеров класса  $c$ .

В нашем примере вычислить weighted recall можно с помощью матрицы ошибок:

$$\text{recall}_{\text{weighted}} = \frac{(9 + 0 + 1) \cdot \frac{9}{9+0+1} + (7 + 1 + 2) \cdot \frac{7}{7+1+2} + (7 + 1 + 2) \cdot \frac{7}{7+1+2}}{(9 + 0 + 1) + (7 + 1 + 2) + (7 + 1 + 2)} = 23/30 = 0.766$$

### F1-мера для многоклассовой классификации

Для одного класса  $c$  в задаче многоклассовой классификации F1-мера вычисляется по формуле 2.18:

$$F1_c = \frac{2 \cdot TP_c}{2 \cdot TP_c + FP_c + FN_c},$$

где  $TP_c$  - количество верно предсказанных положительных примеров класса  $c$ , (2.18)

$FP_c$  - количество ложно предсказанных положительных примеров класса  $c$ ,

$FN_c$  - количество ложно предсказанных отрицательных примеров класса  $c$ .

Здесь F1-мера является гармоническим средним между точностью и полнотой для одного класса.

**Micro F1** - это метрика, которая вычисляет F1-меру для каждого класса по отдельности, а затем усредняет их с использованием общего числа истинных и ложных положительных и отрицательных ответов по всем классам. Эта метрика подходит для задач, где классы не сбалансированы и модель должна давать одинаковый вес каждому объекту. Формула для вычисления micro F1 (Формула 2.19):

$$F1_{\text{micro}} = 2 \cdot \frac{\sum_c TP_c}{\sum_c (TP_c + FP_c) + \sum_c (TP_c + FN_c)} (2.19)$$

В нашем примере вычислить micro F1 можно с помощью матрицы ошибок:

$$F1_{\text{micro}} = 2 \cdot \frac{(9 + 7 + 7)}{11 + 9 + 10 + 10 + 10 + 10} = 23/30 = 0.766$$

**Macro F1** - это метрика, которая вычисляет F1-меру для каждого класса по отдельности и затем усредняет их. Эта метрика не учитывает размеры классов, поэтому она подходит для задач, где классы сбалансированы. Формула для вычисления macro F1 (Формула 2.20):

$$F1_{macro} = \frac{1}{C} * \sum_c F1_c \quad (2.20)$$

где  $C$  - количество классов,  $F1_c$  - F1-мера для класса  $c$

В нашем примере вычислить macro F1 можно с помощью матрицы ошибок:

$$F1_{macro} = \frac{1}{3} * \left( \frac{2 \cdot 9}{2 \cdot 9 + 2 + 1} + \frac{2 \cdot 7}{2 \cdot 7 + 2 + 3} + \frac{2 \cdot 7}{2 \cdot 7 + 3 + 3} \right) = 0.764$$

**Weighted F1** - это метрика, которая вычисляет F1-меру для каждого класса по отдельности и затем усредняет их с использованием весов, пропорциональных размеру каждого класса. Эта метрика подходит для задач, где классы не сбалансированы, и модель должна учитывать важность каждого класса. Формула для вычисления weighted F1 (Формула 2.21):

$$F1_{weighted} = \frac{\sum_c w_c \cdot F1_c}{N} \quad (2.21)$$

где  $N$  - количество примеров во всей выборке,

$w_c$  - доля примеров класса  $c$  в выборке.

В нашем примере вычислить weighted F1 можно с помощью матрицы ошибок:

$$F1_{weighted} = \frac{1}{3} * \left( \frac{2 \cdot 9}{2 \cdot 9 + 2 + 1} + \frac{2 \cdot 7}{2 \cdot 7 + 2 + 3} + \frac{2 \cdot 7}{2 \cdot 7 + 3 + 3} \right) = 0.764$$

### 2.3.3 Обобщающая способность алгоритмов и переобучение

Метрики качества - это не единственное, что нужно учитывать при оценке работы модели машинного обучения. Еще одной важной характеристикой модели является обобщающая способность. Обобщающая способность алгоритма машинного обучения - это способность алгоритма правильно классифицировать новые, ранее неизвестные данные, которые не использовались при обучении. Она является ключевым показателем качества алгоритма машинного обучения.

**Переобучение (англ. Overfitting)** - это явление, когда алгоритм машинного обучения получает очень высокую точность на данных, которые использовались для обучения, но плохо работает на новых данных. При переобучении алгоритм машинного обучения настраивается на шум в данных и пытается «запомнить» обучающие данные, а не выявлять общие закономерности. При переобучении график метрик на тренировочном наборе данных будет показывать улучшение в процессе обучения, но на тестовом наборе данных график метрик может начать падать после достижения пика или перестать улучшаться, что будет указывать на то, что модель не обобщает знания на новые данные (Рисунок 2.8).

Переобучение может возникнуть, если модель слишком сложная и содержит слишком много параметров, которые могут быть настроены на шум в данных. Также переобучение может возникнуть, если обучающая выборка слишком мала или не достаточно разнообразна.

Для предотвращения переобучения существует несколько методов, основным из которых является регуляризация. Мы поговорим о ней в следующем модуле.

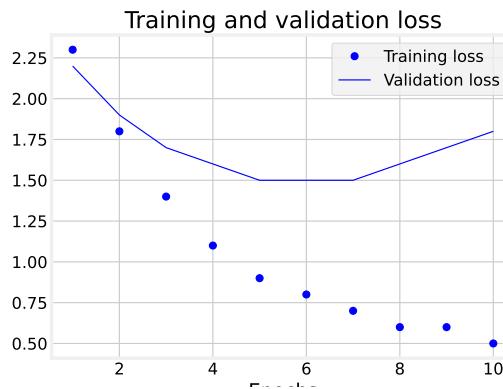


Рисунок 2.8: Пример ситуации переобучения

### 2.3.4 Вопросы для самопроверки

Выберите метрики качества регрессии из приведенных ниже вариантов:

1. accuracy
2. MAE
3. Huber loss
4. MSE

Правильные ответы: 2, 3, 4

Метрика precision - это отношение:

1.  $\frac{TP}{TP+FP}$
2.  $\frac{TP}{TP+FN}$
3.  $\frac{TP+TN}{TP+TN+FP+FN}$

Правильные ответы: 1

Какую модификацию метрики F1 для многоклассовой классификации нужно выбрать, чтобы учесть размеры (баланс) классов:

1. Micro
2. Macro
3. Weighted

Правильные ответы: 3

### 2.3.5 Резюме по разделу

Существует несколько метрик качества, которые используются для оценки качества алгоритмов машинного обучения.

Распространенные метрики качества для задач регрессии:

- Средняя квадратичная ошибка (MSE)
- Средняя абсолютная ошибка (MAE)

- Функция потерь Хьюбера

Распространенные метрики качества для задач классификации:

- Точность (Accuracy)
- Точность положительного класса (Precision)
- Полнота (Recall)
- F1-мера (F1-score)

Выбор метрик зависит от конкретной задачи и ее целей.

Обобщающая способность алгоритмов машинного обучения означает их способность обобщать знания из тренировочных данных и применять их для эффективного решения новых задач на новых данных. То есть, хорошо обученная модель должна показывать высокое качество предсказаний не только на тренировочных данных, но и на тестовых и реальных данных.

Однако, при обучении модели можно столкнуться с проблемой переобучения, когда модель слишком точно подстроилась под тренировочные данные, и, соответственно, ее способность к обобщению становится низкой. В результате, такая модель показывает плохие результаты на тестовых данных и не может быть использована для решения новых задач.

Для того, чтобы избежать переобучения, необходимо использовать специальные техники, такие как регуляризация.

## 2.4 Метод ближайших соседей

**Цель занятия:** ученик может применить алгоритм KNN для решения задач регрессии и классификации на подготовленных и неподготовленных данных.

**План занятия:**

- Визуальная демонстрация алгоритма
- Описание алгоритма
- Подготовка данных для алгоритма
- Оценка качества алгоритма
- Модификации алгоритма
- Область применения алгоритма
- Плюсы и минусы алгоритма
- Реализация алгоритма в Python

### 2.4.1 Визуальная демонстрация алгоритма

Визуальная демонстрация алгоритма представлена на последовательности из 4 рисунков (Рисунок 2.9).

### 2.4.2 Описание алгоритма

**Метод ближайших соседей (англ. KNN, k-Nearest Neighbors)** - это алгоритм машинного обучения для классификации и регрессии. Он относится к методам обучения с учителем, что означает, что для обучения необходимы данные с известными метками классов или целевыми значениями.

Пошаговое описание алгоритма KNN:

1. Загрузка данных. Загрузите обучающий набор данных, содержащий признаки и соответствующие метки классов (для задачи классификации) или значения целевой переменной (для задачи регрессии). Если признаки имеют разные шкалы или единицы измерения, то целесообразно выполнить нормализацию данных. Например, можно использовать стандартизацию, где каждый признак будет иметь среднее значение 0 и стандартное отклонение 1. Определите количество ближайших соседей  $K$ , которые будут использованы для классификации или регрессии. Значение  $K$  должно быть положительным целым числом.
2. Вычисление расстояний. Для каждого экземпляра тестовых данных вычислите расстояние до каждого экземпляра обучающего набора. Расстояние может быть вычислено с использованием различных метрик, таких как евклидово расстояние или манхэттенское расстояние.
3. Сортировка и выбор  $K$  ближайших соседей. Отсортируйте расстояния в порядке возрастания.
4. Выбор  $K$  соседей. Выберите  $K$  ближайших соседей для каждого экземпляра тестовых данных. В случае классификации можно определить метку класса путем выбора наиболее часто встречающейся метки среди  $k$  ближайших соседей. В случае регрессии можно предсказать значение целевой переменной путем вычисления среднего или медианы значений целевой переменной среди  $k$  ближайших соседей.

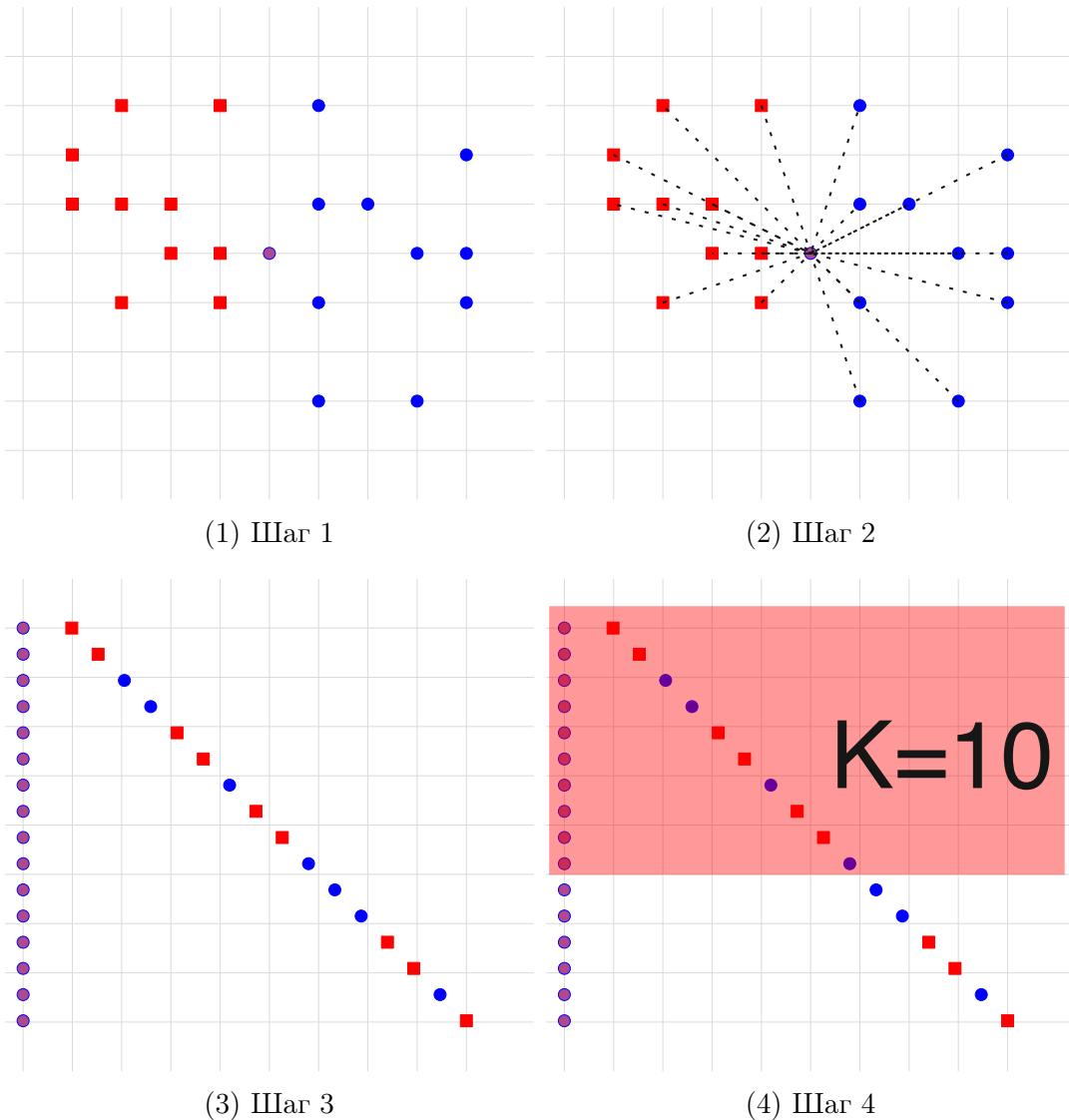


Рисунок 2.9: Визуальная демонстрация алгоритма KNN

Алгоритм k-ближайших соседей (KNN) - это простой алгоритм машинного обучения, который не имеет много параметров обучения. Главный параметр - **число соседей (K)**, которое необходимо выбрать для каждого объекта, чтобы выполнить классификацию или регрессию.

Выбор числа соседей  $K$  является важным шагом в использовании алгоритма KNN. Если выбрать слишком маленькое значение  $K$ , то модель будет слишком чувствительной к выбросам и шуму, что может привести к переобучению. Если выбрать слишком большое значение  $K$ , то модель может потерять способность к выделению малозначимых признаков и потерять способность к различению классов.

Другие параметры, которые могут быть учтены при использовании алгоритма KNN, включают в себя: **меру расстояния** (например, евклидово расстояние или манхэттенское расстояние), **веса для каждого соседа** (например, обратная пропорциональность расстояния до соседа).

Важно отметить, что параметры меры расстояния и веса соседей могут сильно влиять на качество модели, и их выбор должен быть основан на особенностях конкретной задачи. Как правило, оптимальные значения этих параметров подбираются методом кросс-валидации на обучающем наборе данных.

### 2.4.3 Подготовка данных для алгоритма

Перед применением алгоритма KNN необходимо выполнить следующие шаги подготовки данных:

- **Очистка данных:** удаление неполных или некорректных записей, заполнение пропущенных значений и преобразование данных в формат, подходящий для анализа.
- **Нормализация данных:** приведение данных к общему масштабу, чтобы каждый признак имел одинаковый вклад в анализ.
- **Разбиение данных на обучающую и тестовую выборки:** обучающая выборка используется для обучения модели, а тестовая выборка - для оценки ее точности.
- **Выбор метрики расстояния:** выбор подходящей метрики расстояния, которая будет использоваться для определения ближайших соседей.
- **Определение параметра k:** выбор оптимального значения параметра  $k$  (количество ближайших соседей), который дает наилучшие результаты для конкретной задачи, с помощью алгоритмов перебора.

### 2.4.4 Оценка качества алгоритма

Для оценки точности алгоритма KNN в задачах классификации и регрессии часто используют следующие метрики:

#### Для задач классификации:

- **Accuracy (точность):** доля правильно классифицированных объектов.
- **Precision (точность):** доля объектов, классифицированных как положительные и действительно являющихся положительными.
- **Recall (полнота):** доля положительных объектов, правильно классифицированных алгоритмом.
- **F1-мера:** среднее гармоническое между precision и recall.

#### Для задач регрессии:

- **Mean Absolute Error (MAE):** средняя абсолютная ошибка между прогнозами и истинными значениями.
- **Mean Squared Error (MSE):** средняя квадратичная ошибка между прогнозами и истинными значениями.

### 2.4.5 Модификации алгоритма

Существует несколько модификаций алгоритма k-ближайших соседей (KNN). Рассмотрим не так часто используемые на практике, и приведены здесь для ознакомления.

- **Взвешенный KNN (англ. Weighted KNN):** в этой модификации алгоритма каждый сосед получает вес, зависящий от расстояния до целевого объекта. Чем ближе сосед, тем

больший вес он получает. Это позволяет учесть вклад каждого соседа в классификацию или регрессию, и улучшает точность модели.

- **Алгоритм KNN с переменным числом соседей (англ. Variable KNN):** в этой модификации число соседей для каждого объекта может быть разным. Например, для объектов, находящихся в области с большой плотностью точек, можно выбирать большее число соседей, а для объектов в области с малой плотностью - меньшее число соседей. Это позволяет лучше адаптировать модель к особенностям данных.
- **Алгоритм KNN с оптимизацией расстояния (англ. Distance-optimized KNN):** в этой модификации используется не единственная мера расстояния, а оптимизированная комбинация нескольких мер расстояния. Это может улучшить точность классификации или регрессии в задачах с нелинейными зависимостями между признаками.
- **Многоуровневый KNN (англ. Multi-level KNN):** в этой модификации используется несколько моделей KNN на разных уровнях анализа данных. Например, для текстовых данных можно использовать первый уровень KNN для выбора близких по смыслу документов, а затем второй уровень KNN для классификации на основе дополнительных признаков.
- **Алгоритм KNN на основе графов (англ. Graph-based KNN):** в этой модификации алгоритма объекты представляются в виде вершин графа, а соседство между объектами определяется на основе связей в графе. Это позволяет учитывать не только расстояние между объектами, но и их близость в структуре данных.

Каждая из этих модификаций KNN может быть полезной в определенных задачах машинного обучения, и выбор модификации должен быть основан на особенностях данных и конкретной задаче.

#### 2.4.6 Область применения алгоритма

Область применения алгоритма KNN включает в себя многие задачи машинного обучения, такие как:

- Обработка табличных данных.
- Классификация текстов: KNN может использоваться для классификации текстовых документов по их содержанию. Например, можно классифицировать новости на категории на основе их содержания.
- Обработка изображений: KNN может использоваться для классификации изображений на основе их содержания. Например, можно классифицировать изображения по типу объекта на фотографии (человек, автомобиль, животное и т.д.).
- Медицинская диагностика: KNN может использоваться для диагностики заболеваний на основе симптомов и медицинских показателей. Например, можно классифицировать пациентов на основе их медицинской истории.
- Финансовый анализ: KNN может использоваться для анализа финансовых данных и прогнозирования цен на акции, валюту и другие финансовые инструменты.

Кроме того, алгоритм KNN может использоваться во многих других областях, где требуется классификация или регрессия на основе признаков объектов.

#### 2.4.7 Плюсы и минусы алгоритма

Алгоритм k-Nearest Neighbors (KNN) имеет ряд преимуществ и недостатков. **Плюсы:**

- Простота реализации и понимания. Это один из самых простых алгоритмов машинного обучения.
- Может использоваться как для задач классификации, так и для задач регрессии.
- Гибкость в выборе метрики расстояния и числа соседей.

#### Минусы:

- Низкая эффективность на больших выборках и большое время выполнения.
- Чувствительность к выбросам и шуму в данных.
- Требует хранения всего набора данных в памяти, что может быть проблематично для больших наборов данных.
- Трудности с выбором оптимального числа соседей, которые влияют на качество предсказаний.

В целом, алгоритм KNN имеет простую и интуитивно понятную логику, но его эффективность может быть низкой в случае больших выборок и шумных данных.

#### 2.4.8 Реализация алгоритма в Python

В библиотеке Scikit-learn в модуле neighbors реализован класс **KNeighborsClassifier** и **KNeighborsRegressor** для классификации и регрессии соответственно.

Класс KNeighborsClassifier имеет следующие параметры:

- **n\_neighbors**: количество соседей, используемых для классификации объекта. По умолчанию равно 5.
- **weights**: весовая функция, используемая для определения влияния каждого соседа. По умолчанию все соседи имеют равный вес.
- **algorithm**: алгоритм, используемый для вычисления ближайших соседей. Возможные значения: «auto», «ball\_tree», «kd\_tree», «brute». По умолчанию используется «auto», который выбирает наиболее подходящий алгоритм на основе обучающих данных.
- **leaf\_size**: размер листа дерева, используемый при использовании алгоритмов ball\_tree или kd\_tree. По умолчанию равен 30.
- **p**: параметр, используемый для расчета расстояния между объектами. По умолчанию равен 2 (Евклидово расстояние).
- **metric**: метрика расстояния, используемая для измерения расстояния между объектами. По умолчанию используется «minkowski».
- 

Класс KNeighborsRegressor имеет те же параметры, что и KNeighborsClassifier, за исключением параметра weights, который по умолчанию установлен в «uniform».

Оба класса имеют методы **fit(X, y)** для обучения модели на данных X и y, а также методы **predict(X)** и **predict\_proba(X)** (для KNeighborsClassifier) для предсказания меток и вероятностей соответственно. Кроме того, для KNeighborsRegressor есть методы **score(X, y)** и **kneighbors(X)**, которые возвращают коэффициент детерминации (R-квадрат) и индексы ближайших соседей для каждого объекта в X соответственно.

#### 2.4.9 Вопросы для самопроверки

Как происходит обучение алгоритма KNN:

1. алгоритм просто запоминает обучающую выборку, чтобы затем измерить расстояние от нового объекта до объектов в обучающей выборке
2. с помощью градиентного спуска
3. с помощью метода Ньютона

Правильные ответы: 1

Как вычисляется значение для нового объекта в задаче регрессии с помощью KNN:

1. Объекту присваивается значение наиболее близкого объекта из обучающей выборки
2. Объекту присваивается среднее значение по  $k$  наиболее близким объектам из обучающей выборки
3. Объекту присваивается среднее значение по 3 наиболее близким объектам из обучающей выборки

Правильные ответы: 2

Выберите плюсы алгоритма KNN:

1. Гибкость в выборе метрики расстояния и числа соседей
2. Низкая эффективность на больших выборках и большое время выполнения
3. Чувствительность к выбросам и шуму в данных
4. Простота реализации и понимания
5. Требует хранения всего набора данных в памяти, что может быть проблематично для больших наборов данных
6. Трудности с выбором оптимального числа соседей, которые влияют на качество предсказаний

Правильные ответы: 1, 4

#### 2.4.10 Резюме по разделу

**Алгоритм k-Nearest Neighbors (KNN)** - это простой алгоритм машинного обучения, который используется для классификации и регрессии. Алгоритм определяет класс нового объекта или его значение, основываясь на  $k$  ближайших объектах из тренировочного набора данных.

**Параметр  $k$**  - это количество ближайших объектов, которые используются для определения класса нового объекта. Чем больше  $k$ , тем более гладким будет граница принятия решений, но при этом увеличивается шанс на ошибку из-за увеличения влияния объектов других классов.

**Основными преимуществами KNN являются простота реализации** и хорошее качество классификации для некоторых типов данных. Недостатками алгоритма являются высокая вычислительная сложность и требование к наличию большого объема данных для получения высокой точности классификации или регрессии.

Также существуют модификации алгоритма, такие как взвешенный KNN, KNN с использованием ядра и метрики расстояния, а также с использованием алгоритмов уменьшения размерности, которые могут помочь улучшить качество классификации и регрессии с помощью KNN.

## 2.5 Резюме по модулю

В этом модуле мы рассмотрели такие темы, как процесс разработки проекта машинного обучения, подготовка данных для задач машинного обучения, оценка качества алгоритмов обучения с учителем, а также познакомились с одним из самых простых и интуитивно понятных алгоритмов в машинном обучении - методом ближайших соседей.

# Глава 3

## Градиентный спуск для обучения дифференцируемых моделей машинного обучения. Методы регуляризации моделей машинного обучения. Линейная регрессия.

### 3.1 Градиентный спуск

Мы уже познакомились с таким методом машинного обучения, как метод ближайших соседей. Это очень простой метод, обучение в котором, по сути, сводится к запоминанию обучающей выборки. На этом занятии мы приступим к изучению дифференцируемых моделей линейной регрессии, которые обучаются с помощью градиентного спуска. Давайте разберемся, что это такое.

Градиентный спуск - это метод, который помогает модели машинного обучения находить оптимальные значения параметров. Он делает это путем постепенного изменения параметров таким образом, чтобы функция потерь (разница между предсказанными значениями и реальными значениями целевой переменной), становилась все меньше и меньше. Градиентный спуск позволяет найти оптимальные значения параметров модели, минимизируя функцию потерь. Благодаря этому методу можно обучать различные модели машинного обучения, такие как линейные модели, нейронные сети и другие.

Теперь рассмотрим этот процесс подробнее.

#### 3.1.1 Функции потерь и функционалы ошибки в регрессии

И начнем мы наше знакомство с градиентным спуском с таких понятий, как функции потерь и функционалы ошибок в задачах регрессии.

**Функция потерь (loss function)** в задачах регрессии используется для оценки того, насколько хорошо модель предсказывает целевую переменную. Функция потерь выражает разницу между предсказанными значениями и реальными значениями целевой переменной на конкретном обучающем примере.

В задачах регрессии наиболее популярные функции потерь — это квадратичная ошибка, абсолютная ошибка и функция потерь Хьюбера (Рисунок 3.1).

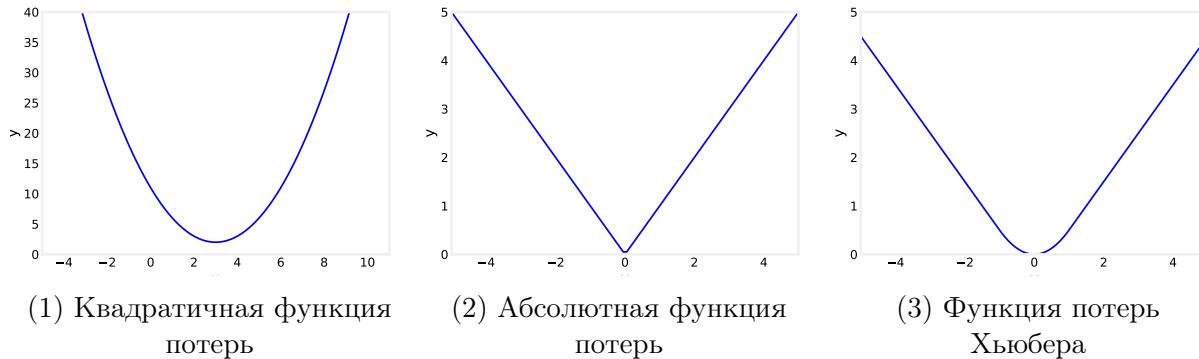


Рисунок 3.1: Популярные функции потерь в задачах регрессии

Функции потерь используются при обучении моделей для определения оптимальных значений параметров модели, которые минимизируют потери. Для определения минимума функции удобно использовать производную функции (Рисунок 3.2).

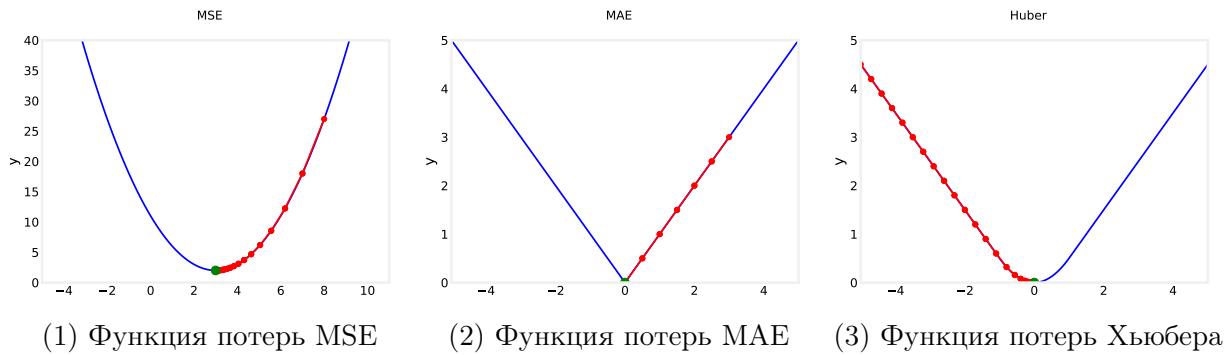


Рисунок 3.2: Нахождение минимума функций с помощью производной

На рисунке 3.2 мы стартуем из произвольной точки, вычисляем в ней производную и движемся в сторону, обратную значению производной, со скоростью, указанной в параметре «learning rate». Траектория движения указана красными линиями. Например, для направленной вверх параболы (квадратичная функция), которая является гладкой функцией с одним глобальным минимумом, производная указывает направление роста функции, соответственно, двигаясь в обратном направлении, можно достичь глобального минимума. Это справедливо и для абсолютной функции потерь. Однако стоит заметить, что в абсолютной функции потерь производная не определена в нуле. Одним из способов решения этой проблемы является использование функции потерь Хьюбера.

Давайте введем еще одно понятие — функционал ошибки. **Функционал ошибки (cost function)** — это математическая функция, которая измеряет среднюю ошибку модели на всем наборе данных. Функционал ошибки представляет собой сумму функции потерь на всех примерах обучающих данных, нормализованную на количество примеров в наборе данных. Функ-

ционал ошибки показывает, как хорошо модель работает на всех данных в целом и служит основной метрикой для оценки качества модели.

Основное отличие между функцией потерь и функционалом ошибки заключается в том, что функция потерь измеряет ошибку на каждом примере данных, в то время как функционал ошибки измеряет среднюю ошибку на всем наборе данных. В задачах регрессии наиболее популярные функционалы потерь — это среднеквадратичная ошибка (MSE), средняя абсолютная ошибка (MAE) и функционал ошибки Хьюбера.

### 3.1.2 Градиент функции потерь

Для минимизации функции потерь, включающей в себя более чем один параметр, в машинном обучении используется градиент — вектор частных производных, который указывает направление наискорейшего возрастания функции потерь относительно параметров модели. Он используется для обновления параметров модели во время обучения с помощью метода градиентного спуска. Градиент функции потерь  $L(\mathbf{x} \cdot \mathbf{w}, \hat{\mathbf{y}})$  обозначается с помощью символа «набла»:  $\nabla L(\mathbf{x} \cdot \mathbf{w}, \hat{\mathbf{y}})$ .

Градиент в машинном обучении может быть вычислен с помощью алгоритмов дифференцирования, таких как автоматическое дифференцирование или символьное дифференцирование. Важно отметить, что в некоторых случаях функция потерь может быть не дифференцируемой или иметь разрывы, что усложняет вычисление градиента и требует использования альтернативных методов оптимизации. В регрессии, например, абсолютная функция потерь недифференцируема в нуле. Для решения этой проблемы используют функцию потерь Хьюбера, где для малых значений функция потерь вычисляется через квадратичную ошибку, а для значений, больших  $\delta$  — через абсолютную ошибку. В функции потерь Хьюбера существует гиперпараметр  $\delta$ , отвечающий за переход от подсчета квадратичной ошибки к абсолютной. Часто в литературе встречается значение  $\delta = 1$ . Но это значение не является универсальным, и его следует выбирать, исходя из характеристик данных и требуемой чувствительности к выбросам. Градиент — это вектор, показывающий направление наискорейшего возрастания функции в заданной точке. В машинном обучении градиент используется для настройки параметров модели с помощью методов оптимизации, таких как градиентный спуск или его модификации.

Некоторые из свойств градиента в машинном обучении:

- С помощью градиента можно находить точки экстремума функции (локальные минимумы и максимумы). Экстремум функции — это точка на ее графике, в которой функция достигает наибольшего (максимум) или наименьшего (минимум) значения в данной области определения (Рисунок 3.3). Найдение локальных и глобального минимумов функции потерь является основной целью использования градиента в машинном обучении. Если функция дифференцируема в экстремуме, то градиент в этой точке равен нулю. Соответственно, с помощью градиента можно искать экстремумы функции. Можно делать это аналитически. Однако в случае сложных функций (Рисунок 3.4), какими зачастую являются функции потерь в машинном обучении, используются численные методы нахождения минимальных значений функции.
- Направление наискорейшего убывания: градиент показывает направление наибольшего увеличения функции в заданной точке. Это позволяет использовать градиент для настройки параметров модели с целью уменьшения функционала ошибки. Для этого используется антиградиент — значение градиента с обратным знаком (Рисунок 3.5).

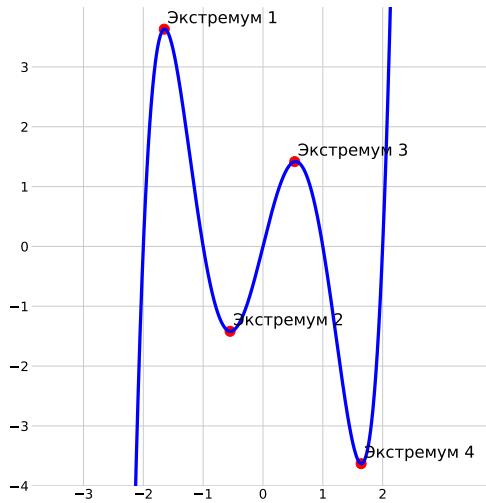


Рисунок 3.3: Экстремумы функции

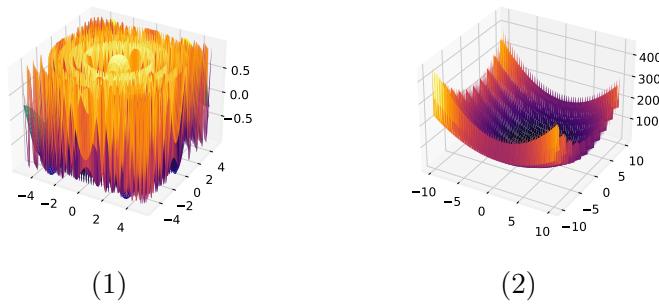


Рисунок 3.4: Сложные функции потерь

Для объяснения использования градиента для выявления направления наискорейшего убывания обычно пользуются метафорой горы. Представьте себе, что вы стоите на вершине горы и хотите спуститься в долину. Вы хотите найти точку с наименьшей высотой долины и двигаться в этом направлении. Вы можете смотреть в любом направлении и определять, куда надо двигаться, чтобы спуститься вниз. Градиент функции — это инструмент, который позволяет понять, в каком направлении двигаться для достижения минимального значения функции. Если градиент функции большой, это значит, что находитесь на крутом склоне горы, и вам нужно двигаться быстрее, чтобы спуститься вниз. Если градиент маленький, это значит, что вы находитесь на пологом участке горы, и двигаться можно медленно (Рисунок 3.6).

- Норма градиента связана со скоростью обучения: чем больше градиент, тем быстрее модель обучается. Однако, слишком большой градиент может привести к неустойчивости процесса оптимизации и переобучению модели.
- Градиент может быть вычислен аналитически: в некоторых случаях, градиент функции может быть выражен в явном виде, что упрощает процесс оптимизации. Например, для

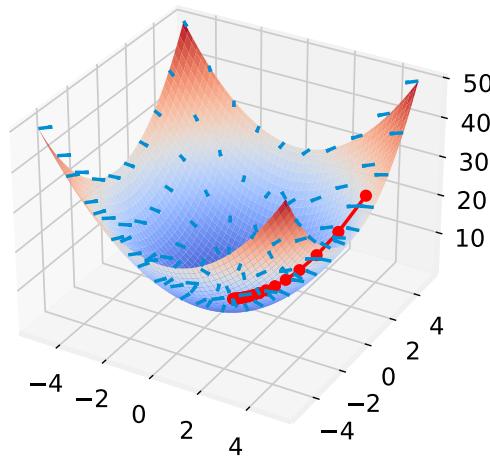


Рисунок 3.5: Использование антиградиента для нахождения минимума функции



Рисунок 3.6: Аналогия градиента и спуска с горы

линейной регрессии, градиент можно выразить в явном виде в случае использования среднеквадратичной ошибки в качестве функционала ошибки.

- Градиент может быть вычислен численно: в случаях, когда аналитический градиент не может быть вычислен, можно использовать численное дифференцирование для приближенного вычисления градиента.
- Градиент направлен перпендикулярно линии уровня в определенной точке функции. Линия уровня функции - это множество точек в ее области определения, в которых значение функции остается постоянным, т.е. все точки на линии имеют одинаковое значение функции (Рисунок 3.7).

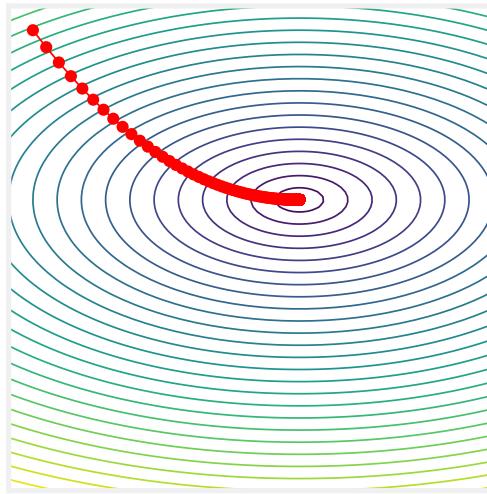


Рисунок 3.7: Градиент перпендикулярно линии уровня

### 3.1.3 Определение градиентного спуска

Теперь мы познакомились со всеми понятиями, необходимыми для определения градиентного спуска. Градиентный спуск - это метод оптимизации, используемый в машинном обучении для нахождения локального минимума функции потерь модели. Он основан на вычислении градиента функции потерь по параметрам модели и последующем обновлении этих параметров в направлении наиболее быстрого убывания градиента.

Псевдокод алгоритма градиентного спуска описан ниже.

**Входные параметры:** Функция потерь  $L(w)$ , начальные параметры  $w_0$ , шаг обучения  $\alpha$ , максимальное число итераций  $T$ , допустимая ошибка  $\epsilon$

**Выходные параметры:** Оптимальные параметры  $w^*$

```

1:  $w \leftarrow w_0$  // Инициализируем значения весов  $w$  с помощью  $w_0$ 
2:  $t \leftarrow 0$  // Инициализируем счетчик итераций  $t$ 
3:  $e \leftarrow \infty$  // Инициализируем значение функции потерь  $\infty$ 
4: Запускаем цикл while
4: while  $t < T$  do // Пока совершено менее  $T$  шагов
4:   Вычислить функцию потерь  $L(w_t)$  и градиент  $\nabla L(w_t)$ 
4:   Обновить параметры:  $w_{t+1} \leftarrow w_t - \alpha_t \nabla L(w_t)$  //  $\alpha_t$  — скорость обучения на текущем
      шаге
      // Проверить выполнение критерия останова, в данном случае, если норма разности но-
      вого и старого векторов весов  $< \epsilon$ 
4:   if  $\|w_{t+1} - w_t\| < \epsilon$  then
4:     break
4:   end if
4:    $t \leftarrow t + 1$ 
4: end while
5: return  $w^* = 0$ 
```

Градиентный спуск является одним из основных методов оптимизации в машинном обучении, который может использоваться для настройки параметров любых моделей, которые могут быть обучены с помощью градиентного спуска.

### 3.1.4 Параметры градиентного спуска

Параметры градиентного спуска включают:

- Скорость обучения (learning rate): это гиперпараметр, который определяет размер шага в каждой итерации обновления параметров. Если скорость обучения выбрана слишком большой, то метод может не сойтись, и оптимизация не будет достигнута. Если скорость обучения слишком мала, метод может занять слишком много времени на поиск оптимума. Выбор оптимальной длины шага (шага градиентного спуска) в градиентном спуске - это важный вопрос, который может существенно влиять на скорость и точность сходимости алгоритма. Вот несколько способов выбора длины шага в градиентном спуске:
  - Адаптивный шаг (adaptive step): длина шага изменяется по формуле или в зависимости от ситуации на каждой итерации. Например, можно использовать методы, такие как AdaGrad, RMSprop или Adam, которые адаптивно изменяют длину шага в зависимости от градиента и предыдущих шагов. Приведем пример базовой формулы для изменения скорости обучения (Формула 3.1):

$$\alpha_t = \alpha_0 \cdot \frac{1}{1 + d \cdot t},$$

где  $\alpha_t$  - скорость обучения в момент времени  $t$ ,  
 $\alpha_0$  - начальная скорость обучения,  
 $d$  - коэффициент затухания скорости обучения,  
 $t$  - текущая эпоха обучения.

- Фиксированный шаг (constant step): длина шага остается постоянной в течение всего процесса обучения. Этот метод прост в реализации, но может быть неэффективным, так как шаг может быть слишком маленьким или большим, что приведет к медленной сходимости или неустойчивости.
- Backtracking line search: это метод, который на каждой итерации градиентного спуска ищет оптимальную длину шага, которая удовлетворяет определенным критериям. Например, можно использовать метод Армихо (Armijo), который гарантирует уменьшение функции потерь на каждом шаге, или метод золотого сечения (golden section), который находит оптимальную длину шага в заданном интервале.
- Функция потерь (loss function): это функция, которая измеряет разницу между текущими предсказаниями и правильными ответами. Оптимизационный алгоритм использует градиент функции потерь для определения направления, в котором нужно обновлять параметры.
- Метод инициализации параметров: это способ задания начальных значений параметров. Он может влиять на скорость сходимости метода, так как некоторые методы инициализации могут быть более эффективными, чем другие. Некоторые из наиболее распространенных методов инициализации включают в себя:

- Случайная инициализация - начальные значения параметров модели выбираются случайным образом. Это является наиболее распространенным методом инициализации.
- Инициализация нулями - начальные значения всех параметров устанавливаются в ноль. Этот метод может быть полезным, если модель уже имеет предварительно обученные веса, которые затем можно дообучить на новых данных.
- Инициализация на основе распределения - начальные значения параметров выбираются из определенного распределения, такого как нормальное или равномерное. Этот метод может быть полезным для определенных типов моделей и задач.

Если начальные значения слишком далеки от оптимальных, то метод может потребовать больше времени на достижение оптимума. Для решения проблемы застrevания в неоптимальных локальных минимумах существует техника «мультистарта». Мультистарт (англ. «multistart») - это метод инициализации весов модели, который заключается в запуске обучения модели несколько раз с разными начальными значениями весов. Использование мультистарта может улучшить качество обучения модели и уменьшить вероятность застrevания в локальных оптимумах.

- Количество итераций: это количество шагов обновления параметров, которые нужно выполнить, чтобы достичь оптимума. Это может быть фиксированный параметр или зависеть от критерия сходимости.
- Критерии остановки: критерии остановки градиентного спуска определяют, когда следует остановить итерации алгоритма оптимизации. Это важно для того, чтобы избежать бесконечного цикла и переобучения модели. Вот некоторые распространенные критерии остановки градиентного спуска:

- Достижение определенного количества итераций. Задается максимальное число итераций, после чего алгоритм останавливается. Количество итераций, необходимых для обучения алгоритма машинного обучения, может значительно различаться в зависимости от многих факторов, таких как размер и сложность набора данных, выбранный алгоритм обучения, используемая аппаратная конфигурация, настройки гиперпараметров и т.д.

В целом, обучение алгоритма машинного обучения требует проведения множества итераций или эпох, где одна эпоха представляет собой один проход по всем обучающим примерам в наборе данных. Часто требуется проводить многократные эпохи для достижения оптимальных результатов обучения.

Например, для некоторых типов нейронных сетей на больших наборах данных, может потребоваться проведение сотен или даже тысяч итераций, прежде чем модель сможет достигнуть высокой точности предсказаний. Однако, для более простых моделей или на более маленьких наборах данных, может потребоваться гораздо меньшее количество итераций (до 100) для достижения приемлемых результатов.

- Достижение определенной точности. Определяется, какой порог разницы между значениями функции потерь на текущей и предыдущей итерациях считать достаточным для остановки алгоритма. Если разница меньше порога, то алгоритм прекращает работу.
- Норма градиента становится меньше заданного значения. Норма градиента определяет скорость изменения функции потерь в каждой точке пространства параметров модели. Если норма градиента становится меньше заданного значения, то это означает, что модель достигла локального минимума, и алгоритм может быть остановлен.

- Сходимость функции потерь. Алгоритм останавливается, когда значения функции потерь перестают существенно меняться на протяжении нескольких итераций.
- Проверка изменения параметров модели. Алгоритм может быть остановлен, когда значения параметров модели перестают изменяться существенно на протяжении нескольких итераций.
- Превышение максимального времени работы. Можно установить максимальное время работы алгоритма, после истечения которого он будет остановлен. Время можно установить в зависимости от сложности задачи и располагаемых вычислительных ресурсов.

Выбор критерия остановки градиентного спуска зависит от конкретной задачи и требуемой точности. Хорошей практикой является комбинирование нескольких критериев, чтобы получить более надежный результат, например, одновременное отслеживание нормы градиента, изменения параметров модели, сходимости функции потерь.

### 3.1.5 Модификации градиентного спуска

Существуют различные модификации алгоритма градиентного спуска, которые позволяют улучшить сходимость и скорость работы. Рассмотрим некоторые из них:

#### Модификации с обучением не по всей выборке

Стохастический градиентный спуск (SGD) и мини-батч градиентный спуск (Mini-batch GD) являются методами оптимизации для обучения нейронных сетей и других моделей машинного обучения.

SGD используется для обновления весов модели после каждого примера обучающего набора, в то время как Mini-batch GD используется для обновления весов после каждой небольшой группы примеров, называемой мини-батч.

#### Преимущества модификаций с обучением не по всей выборке

- Они могут быстрее обучать модели, так как обновления происходят на каждом примере/мини-батче.
- Они могут работать с большими обучающими наборами данных, которые не могут поместиться в память компьютера для GD.
- Они могут помочь избежать застревания в локальных оптимумах, поскольку они более случайные, чем обычный градиентный спуск.

#### Формулы модификаций:

- Стохастический градиентный спуск (SGD). Вместо вычисления градиента по всей выборке данных, SGD вычисляет градиент только по одному обучающему объекту на каждой итерации. Это позволяет ускорить обучение и уменьшить требования к памяти.

Формула SGD представлена в уравнении 3.2.

$$w_{t+1} = w_t - \alpha_t \nabla L(w_t; y_{it}, m(x_{it})),$$

где  $w_t$  - вектор параметров модели на шаге  $t$ ,

$\alpha_t$  - скорость обучения (learning rate), определяющая величину шага в каждой итерации,

$\nabla L$  - градиент функции потерь  $L(w_t; y_{it}, m(x_{it}))$  по параметрам модели  $m$ ,

$x_{it}$  и  $y_{it}$  -  $i$ -й обучающий пример из обучающего набора данных,

используемый на шаге  $t$ .

(3.2)

- Мини-пакетный градиентный спуск (Mini-batch GD). Это комбинация стохастического градиентного спуска и полного градиентного спуска. На каждой итерации алгоритм вычисляет градиент на подвыборке данных фиксированного размера (например, 32 или 64 элемента), что позволяет улучшить скорость обучения и сходимость. Формула mini-batch GD представлена в уравнении 3.3.

$$w_{t+1} = w_t - \alpha_t \frac{1}{length(MB)} \sum_{i \in MB} \nabla L(w_t; y_{it}, m(x_{it}))$$

где  $w_t$  - вектор параметров модели на шаге  $t$ ,

$\alpha_t$  - скорость обучения (learning rate), определяющая величину шага в каждой итерации,

$MB$  - mini-batch (мини-пакет) размера  $length(MB)$ , выбранный случайным образом из обучающего набора данных. Обычно  $MB$  выбирается от нескольких десятков до

нескольких сотен обучающих примеров ,

в зависимости от размера обучающего набора данных,

$\nabla L$  - градиент функции потерь  $L(w_t; y_{it}, m(x_{it}))$  по параметрам модели  $m$ ,

$x_{it}$  и  $y_{it}$  -  $i$ -й обучающий пример из mini-batch набора данных.

(3.3)

## Модификации с инерцией

**Градиентный спуск с инерцией (Momentum)** и **Nesterov Accelerated Gradient (NAG)** являются расширениями стандартного градиентного спуска, которые позволяют учитывать различный масштаб признаков и быстрее продвигаться к оптимуму функции потерь со сложными линиями уровней.

Оптимизация с помощью Momentum - это метод стохастической оптимизации, который ускоряет процесс обучения за счет накопления «импульса» градиентов векторов.

В процессе обучения градиенты вычисляются на каждом шаге для обновления параметров модели. При использовании Momentum, градиенты на каждом шаге умножаются на коэффициент «импульса» (обычно примерно равный 0.9), который указывает, какую долю предыдущего шага нужно сохранить при вычислении градиентов на следующем шаге.

Таким образом, импульс помогает ускорить компоненты градиента в тех направлениях, в которых они сохраняются на протяжении нескольких последовательных шагов, и затормозить в направлениях, в которых градиенты меняются быстро. Это позволяет быстрее и более стабильно сходиться к локальному оптимуму и уменьшить вероятность застревания в локальном минимуме.

При использовании метода Momentum также возможно снижение количества осцилляций (взлетов и падений) при движении к минимуму функции потерь.

Метод Momentum можно применять совместно с различными алгоритмами оптимизации, такими как SGD (стохастический градиентный спуск), Adam и RMSprop.

Nesterov Accelerated Gradient (NAG) - это метод стохастической оптимизации, который является модификацией метода Momentum. В методе Momentum градиенты на каждом шаге умножаются на коэффициент «импульса» и складываются с предыдущим шагом для обновления параметров модели. Однако, в этом случае, имеется небольшая задержка в обновлении параметров, так как градиенты вычисляются на текущей позиции, а обновление параметров происходит на следующем шаге.

Метод Nesterov Accelerated Gradient (NAG) позволяет исправить эту проблему, предварительно заглянув в будущее и вычислив градиент на предполагаемой позиции в следующем шаге, а не на текущей позиции, перед вычислением импульса. Это позволяет получить более точный градиент, что в свою очередь уменьшает вероятность перепрыгивания через локальный оптимум при обновлении параметров.

Более конкретно, в методе NAG сначала вычисляются градиенты на текущей позиции, затем делается шаг в направлении, заданном градиентами на следующей позиции, и только затем вычисляются импульсы, которые накапливаются и добавляются к шагу.

#### Преимущества модификаций с инерцией:

- Они помогают избежать локальных минимумов и ускоряют сходимость.
- Они позволяют обучать более глубокие и сложные модели с большим количеством параметров.
- Они устойчивы к застреванию в локальных оптимумах и помогают более быстро достигать глобального минимума функции потерь.

#### Формулы модификаций:

- Моментум (Momentum). Эта модификация градиентного спуска учитывает предыдущие изменения вектора градиента при обновлении весов модели. Это позволяет ускорить обучение и уменьшить вероятность застревания в локальных минимумах.

$$v_{t+1} = \eta v_t + \alpha_t \nabla L(w_t)$$

$$w_{t+1} = w_t - v_{t+1}$$

где  $v_t$  - «momentum», усредненное направление движения на шаге  $t$ ,

$\eta$  - коэффициент момента, обычно выбирается около 0.9 для большинства задач,

$\alpha_t$  - скорость обучения (learning rate), определяющая величину шага в каждой итерации,

$\nabla L$  - градиент функции потерь  $L(w_t)$  по параметрам модели  $t$ ,

$w_t$  - вектор параметров модели на шаге  $t$ .

(3.4)

Формула градиентного спуска с моментом добавляет вектор скорости  $v_t$ , который является экспоненциальным сглаживанием градиента на предыдущих шагах. Это помогает ускорить процесс оптимизации и сгладить колебания, особенно на «шумных» данных.

- Nesterov Accelerated Gradient (NAG). Это модификация градиентного спуска, которая улучшает сходимость за счет добавления момента движения. В отличие от обычного градиентного спуска, который обновляет веса на основе текущего градиента, Nesterov

momentum сначала делает шаг в направлении предыдущего накопленного градиента, а затем корректирует его на основе текущего градиента.

$$v_{t+1} = \eta v_t + \alpha_t \nabla L(w_t - \eta v_t)$$

$$w_{t+1} = w_t - v_{t+1}$$

где  $v_t$  - «Nesterov momentum», усредненное направление движения на шаге  $t$ ,

$\eta$  - коэффициент момента, обычно выбирается около 0.9 для большинства задач,

$\alpha_t$  - скорость обучения (learning rate), определяющая величину шага в каждой итерации,

$\nabla L$  - градиент функции потерь  $L(w_t)$  по параметрам модели  $t$ ,

$w_t$  - вектор параметров модели на шаге  $t$ .

(3.5)

Формула Nesterov Accelerated Gradient добавляет корректировку вектора скорости  $v_t$  перед вычислением градиента. Это позволяет более точно оценить градиент cost функции и ускоряет процесс оптимизации.

### Модификации с адаптивной скоростью изменения каждого веса

Модификации градиентного спуска с адаптивной (независимой для каждого веса) скоростью изменения весов, такие как Adagrad, RMSProp, являются расширениями стандартного градиентного спуска, которые позволяют эффективно оптимизировать функции потерь, учитывая различные характеристики градиента.

Adagrad - это метод стохастической оптимизации, который позволяет автоматически адаптировать скорость обучения для каждого параметра модели в процессе обучения.

Основная идея метода Adagrad заключается в том, чтобы присваивать больший вес параметрам, для которых наблюдается более редкое обновление, и меньший вес параметрам, для которых наблюдается частое обновление. Для этого на каждом шаге метод Adagrad вычисляет диагональную матрицу  $G$ , которая хранит сумму квадратов градиентов для каждого параметра на протяжении всего обучения. Затем скорость обучения для каждого параметра на текущем шаге вычисляется путем деления скорости обучения на корень из суммы квадратов градиентов для данного параметра на предыдущих шагах.

Преимущество метода Adagrad заключается в том, что он позволяет более эффективно обновлять параметры модели для различных направлений в пространстве параметров, что позволяет более быстро и точно сходиться к оптимальному решению. Кроме того, Adagrad не требует подбора скорости обучения, что является важным преимуществом в задачах с большим количеством параметров. Однако, метод Adagrad может иметь проблемы с сходимостью в случае, когда сумма квадратов градиентов становится очень большой. Эта проблема может быть решена с помощью методов оптимизации, основанных на Adagrad, таких как Adadelta и RMSprop.

RMSprop (Root Mean Square Propagation) - это метод стохастической оптимизации, который использует историю градиентов для адаптации скорости обучения на каждом шаге.

В методе RMSprop, вместо хранения суммы квадратов градиентов для каждого параметра, как в Adagrad, используется экспоненциально взвешенное среднее (exponential moving average) для хранения истории градиентов.

Преимущество метода RMSprop заключается в том, что он позволяет более эффективно обновлять параметры модели для различных направлений в пространстве параметров, учитывая историю градиентов. Кроме того, RMSprop автоматически адаптирует скорость обучения для каждого параметра, что упрощает настройку гиперпараметров.

Недостатком метода RMSprop является то, что он не учитывает адаптацию скорости обучения в различных направлениях в пространстве параметров, что может привести к неэффективному движению вдоль осей, где градиенты меньше, чем вдоль осей, где градиенты больше.

#### **Преимущества градиентного спуска с адаптивной скоростью изменения каждого веса**

- Они позволяют быстрее и более эффективно оптимизировать функции потерь, учитывая различные характеристики градиента.
- Они могут помочь избежать застревания в локальных оптимумах и быстрее достигать глобального минимума функции потерь.
- Они устойчивы к изменениям в данных и помогают улучшить обобщающую способность модели.

#### **Формулы модификаций:**

- Адаптивный градиентный спуск (Adaptive GD, AdaGrad). Эта модификация алгоритма позволяет изменять длину шага (learning rate) в зависимости от геометрических свойств поверхности функции потерь. Например, метод Adagrad позволяет изменять длину шага для каждого параметра модели в зависимости от того, как часто он обновляется.

$$\begin{aligned} G_{jt+1} &= G_{jt} + \nabla L(w_t)_j^2 \\ w_{jt+1} &= w_{jt} - \frac{\alpha_t}{\sqrt{G_{jt} + \epsilon}} \nabla L(w_t)_j \end{aligned}$$

$G_{jt}$  - накопленные квадраты  $j$  компонента градиента для шага  $t$ ,

$\nabla L$  - градиент функции потерь  $L(w_t)$  по параметрам модели  $t$ , (3.6)

$\alpha_t$  - скорость обучения (learning rate),

определяющая величину шага в каждой итерации,

$\epsilon$  - значение, добавляемое для стабилизации вычислений,

$w_t$  - вектор параметров модели на шаге  $t$ .

Формула Adagrad предлагает уникальный подход к выбору скорости обучения для каждого параметра, основанный на адаптивном масштабировании скорости обучения в соответствии с суммой квадратов предыдущих градиентов. Adagrad эффективен для работы с разреженными данными и шумными градиентами, что делает его популярным в обработке естественного языка и компьютерном зрении.

- Алгоритм RMSprop (Root Mean Square Propagation). Эта модификация алгоритма корректирует скорость обучения каждого параметра в сети на основе среднего квадрата градиентов для этого параметра. Интуиция за алгоритмом RMSprop заключается в том, чтобы уделять больше внимания последним градиентам и меньше внимания прошлым

градиентам при обновлении параметров модели.

$$\begin{aligned} G_{jt+1} &= \eta G_{jt} + (1 - \eta) \nabla L(w_t)_j^2 \\ w_{jt+1} &= w_{jt} - \frac{\alpha_t}{\sqrt{G_{jt} + \epsilon}} \nabla L(w_t)_j \end{aligned}$$

$G_{jt}$  - накопленные квадраты  $j$  компонента градиента для шага  $t$ ,

$\eta$  - коэффициент затухания (damping factor),

обычно выбирается около 0.9 для большинства задач,

$\nabla L$  - градиент функции потерь  $L(w_t)$  по параметрам модели  $t$ ,

$\alpha_t$  - скорость обучения (learning rate),

определяющая величину шага в каждой итерации,

$\epsilon$  - значение, добавляемое для стабилизации вычислений,

$w_t$  - вектор параметров модели на шаге  $t$ .

Формула RMSprop использует скользящее среднее квадрата градиента, чтобы нормировать скорость обучения для каждого параметра. Это позволяет уменьшить влияние выбросов в градиентах на процесс обучения и улучшить сходимость. Коэффициент затухания позволяет «забывать» более старые значения градиента, чтобы учитывать только более актуальные значения.

### Модификации с инерцией и адаптивной скоростью изменения каждого веса

Adam (Adaptive Moment Estimation) - это метод стохастической оптимизации, который комбинирует идеи методов Momentum и RMSprop. Adam использует экспоненциально взвешенное среднее градиентов, как в RMSprop, и добавляет коррекцию смещения, как в методе Momentum, чтобы обеспечить более стабильную и сбалансированную оптимизацию.

**Преимущества модификации с инерцией и адаптивной скоростью изменения каждого веса:**

- Адаптивная скорость обучения: Adam автоматически адаптирует скорость обучения для каждого параметра в процессе оптимизации, что позволяет более быстро достигнуть оптимального значения функции потерь.
- Контроль момента: Adam использует два экспоненциально слаженных момента градиента для адаптивного выбора скорости обучения. Это позволяет более эффективно обрабатывать шум в данных и делать более точные оценки градиента.
- Эффективность: Adam показывает хорошую производительность на больших наборах данных и быстро сходится к оптимальному значению функции потерь.
- Использование памяти: Adam хранит только первые и вторые моменты градиента, что позволяет оптимизировать параметры на больших наборах данных, не требуя больших объемов оперативной памяти.
- Сходимость: Adam показывает быструю сходимость на многих задачах, и его параметры могут быть настроены автоматически при помощи алгоритмов оптимизации гиперпараметров.

Недостатком метода Adam является его сложность, которая может замедлить процесс обучения на больших моделях. Также в некоторых случаях метод Adam может не сходиться к

оптимальному решению. Для некоторых задач может быть более эффективным использовать методы оптимизации, основанные на методе Momentum или RMSprop.

#### Формула модификации:

- Формула Adam объединяет идеи из Momentum и RMSprop и использует два скользящих средних - скользящее среднее градиента  $m_t$  и скользящее среднее квадрата градиента  $v_t$ . Она исправляет начальное смещение в начале оптимизации и адаптивно регулирует скорость обучения для каждого параметра. Adam является одним из наиболее эффективных методов градиентного спуска и часто используется в глубоком обучении.

$$\begin{aligned} g_t &= \nabla L(w_t) \\ m_t &= \beta_1 m_{t-1} + (1 - \beta_1)g_t \\ v_t &= \beta_2 v_{t-1} + (1 - \beta_2)g_t^2 \\ \hat{m}_t &= \frac{m_t}{1 - \beta_1^t} \\ \hat{v}_t &= \frac{v_t}{1 - \beta_2^t} \\ w_{t+1} &= w_t - \frac{\alpha}{\sqrt{\hat{v}_t} + \epsilon} \hat{m}_t, \end{aligned}$$

где  $g_t$  - градиент на шаге  $t$ ,

$m_t$  - скользящее среднее градиента на шаге  $t$ ,

$\beta_1$  и  $\beta_2$  - коэффициенты момента,

обычно выбираются около 0.9 и 0.999 соответственно,

$v_t$  - скользящее среднее квадрата градиента на шаге  $t$ ,

$\hat{m}_t$  и  $\hat{v}_t$  - исправленные значения скользящего среднего градиента

и квадрата градиента на шаге  $t$ , чтобы учитывать начальное смещение в начале оптимизации,

$v_t$  - скользящее среднее квадрата градиента на шаге  $t$ ,

$\alpha_t$  - скорость обучения (learning rate), определяющая величину шага в каждой итерации,

$\epsilon$  - значение, добавляемое для стабилизации вычислений,

$w_t$  - вектор параметров модели на шаге  $t$ .

(3.8)

### 3.1.6 Вопросы для самопроверки

Чем отличается loss function от cost function в контексте машинного обучения?

- Loss function используется для оценки ошибки на одном примере, а cost function - для оценки ошибки на всем наборе данных.
- Loss function используется для оценки ошибки на всем наборе данных, а cost function - для оценки ошибки на одном примере.
- Loss function и cost function - это одно и то же понятие.
- Loss function и cost function отличаются только в своем названии и используются для оценки ошибки на всем наборе данных.

Правильные ответы: 1

Что происходит на каждой итерации алгоритма градиентного спуска?

1. Вычисляется значение функции потерь на одном случайно выбранном примере данных, затем значения параметров модели обновляются для уменьшения этого значения функции потерь.
2. Вычисляются частные производные функции потерь по всем параметрам модели, затем значения параметров обновляются с использованием этих частных производных.
3. Вычисляется производная функции потерь по одному случайно выбранному параметру модели, затем значение этого параметра обновляется с использованием этой производной.
4. Вычисляется значение функции потерь на всем наборе данных, затем значения параметров модели обновляются для уменьшения этого значения функции потерь.

Правильные ответы: 2

В каких модификациях градиентного спуска обновление весов происходит адаптивно для каждого из весов, с учетом особенностей признаков (3 ответа)?

1. Градиентный спуск с инерцией
2. Nesterov Accelerated Gradient
3. Adagrad
4. RMSProp
5. Adam

Правильные ответы: 3,4,5

### 3.1.7 Резюме по разделу

Функции потерь и функционалы ошибок являются важными инструментами в машинном обучении, так как они позволяют оценивать качество модели и управлять процессом обучения.

Функция потерь - это функция, которая оценивает расхождение между предсказанными значениями модели и фактическими значениями целевой переменной на конкретном примере данных. Функционал ошибки - это функция, которая оценивает общее качество модели на всем наборе данных. Функционал ошибки вычисляется путем усреднения функции потерь по всем примерам в наборе данных. Оптимизация функционала ошибки является целью обучения модели, так как это позволяет создать модель, которая хорошо обобщает данные и способна делать точные прогнозы на новых данных.

Функции потерь и функционалы ошибок используются для настройки параметров модели в процессе обучения. Цель состоит в том, чтобы минимизировать значение функции потерь или функционала ошибки путем обновления параметров модели. Оптимизация функции потерь происходит с помощью градиентного спуска.

Существуют различные модификации алгоритма градиентного спуска:

- Модификации с обучением не по всей выборке (SGD, Mini-batch GD)
- Модификации с инерцией (Градиентный спуск с инерцией, Nesterov Accelerated Gradient)
- Модификации с адаптивной скоростью изменения каждого веса (Adagrad, RMSProp)
- Модификации с инерцией и адаптивной скоростью изменения каждого веса (Adam)

## 3.2 Регуляризация

Мы уже немного затрагивали тему регуляризации в предыдущем модуле. На этом занятии мы более подробно разберем различные методы регуляризации.

Регуляризация в машинном обучении — это методика, которая помогает снизить переобучение моделей. Переобучение возникает, когда модель слишком хорошо подстраивается под тренировочные данные, что приводит к тому, что она плохо обобщается на новые данные. Регуляризация может быть полезна в многих задачах машинного обучения, особенно в случаях, когда у вас мало тренировочных данных или когда модель имеет очень много параметров.

### 3.2.1 Основные методы регуляризации

Существует несколько основных методов регуляризации в машинном обучении, включая:

- L1-регуляризация (Lasso): добавляет к функции потерь модели сумму абсолютных значений весов, что приводит к разреженности весов и выбору наиболее значимых признаков.
- L2-регуляризация (Ridge): добавляет к функции потерь модели квадрат суммы всех весов, что приводит к сокращению весов и снижению влияния шумовых признаков.
- Elastic Net: комбинация L1-регуляризации и L2-регуляризации, которая помогает учесть преимущества обоих методов и уменьшить их недостатки.
- Dropout: случайным образом исключает некоторые узлы из обучения на каждом шаге, что помогает снизить переобучение и повысить обобщающую способность модели (используется в глубинном обучении).
- Data augmentation: методика, которая заключается в создании новых тренировочных данных путем искажения и изменения исходных данных, что помогает снизить переобучение и увеличить обобщающую способность модели.
- Early stopping: методика, которая заключается в прекращении обучения модели, когда ее производительность на проверочном наборе данных перестает улучшаться, что помогает избежать переобучения и выбрать наилучшую модель.

L1-регуляризация, L2-регуляризация и Elastic Net можно использовать с любыми моделями, где оптимизируются веса с помощью функции потерь. Early stopping применяется в случаях, когда происходит мониторинг метрик качества или изменения норм весов. Dropout и Data augmentation применяются в основном в нейросетевых моделях.

### 3.2.2 Методы регуляризации, основанные на добавлении слагаемого к функции потерь модели

Давайте подробнее разберем те типы регуляризации, которые работают путем добавления дополнительного слагаемого к функции потерь модели. Существует два основных типа такой регуляризации: L1-регуляризация и L2-регуляризация. L1-регуляризация добавляет к функции потерь модели сумму абсолютных значений всех весов модели, тогда как L2-регуляризация добавляет к функции потерь модели квадрат суммы всех весов модели. Это дополнительное слагаемое, которое называется регуляризатором, заставляет модель предпочитать более простые решения, которые не зависят слишком сильно от конкретных тренировочных данных. Интуиция за данным действием следующая: большие веса признака свидетельствуют о том, что модель переобучилась. Соответственно, добавление дополнительного

слагаемого к функции потерь заставляет модель снижать значение весов признаков. Принцип регуляризации, основанной на добавлении слагаемого к функции потерь модели, можно продемонстрировать на следующей иллюстрации: с помощью полиномиальной функции 1, 4, и 8 степени мы пытаемся выучить закономерности в данных (Рисунок 3.8).

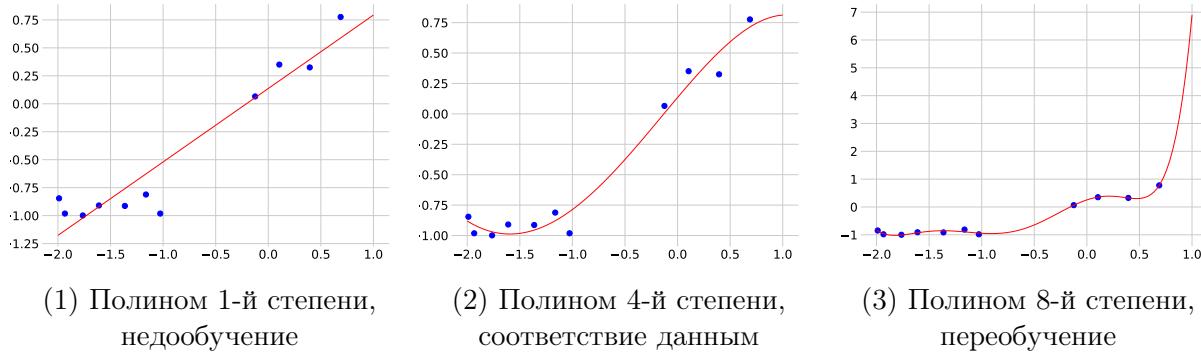


Рисунок 3.8: Иллюстрация обобщающей способности модели машинного обучения

Мы видим, что отсутствие регуляризации (Рисунок 3.8.3) приводит к слишком сильному подстраиванию под данные, что соответствует ситуации переобучения. Наоборот, слишком сильная регуляризация (Рисунок 3.8.1) приводит к упрощению модели и недообучению. Оптимальный вариант показан на рисунке 3.8.2. В этом случае регуляризация оптимальна, модель правильно улавливает закономерности в данных. Сила регуляризации в L1 и L2 регуляризации контролируется через параметр  $\lambda$ .

## L1-регуляризация

L1-регуляризация (или Lasso, от англ. Least Absolute Shrinkage and Selection Operator) - это метод регуляризации в машинном обучении, который использует сумму абсолютных значений весов модели, чтобы ограничить их величину. Он работает путем добавления слагаемого, пропорционального сумме абсолютных значений весов модели, к общей функции потерь модели (Уравнение 3.9).

$$L_{L1} = L + \lambda \sum_{j=1}^d |w_j|, \quad (3.9)$$

где  $L$  - функция потерь без регуляризации,  $\lambda$  - гиперпараметр, который контролирует силу регуляризации,  $d$  - количество признаков,  $w_j$  - вес признака  $j$ .

Плюсы L1-регуляризации:

- Одним из главных преимуществ L1-регуляризации является ее способность к отбору признаков: модель, обученная с использованием L1-регуляризации, часто имеет многие веса, которые равны нулю, что позволяет определить, какие признаки важны для предсказания и какие можно исключить. Это упрощает модель и уменьшает шум в данных, что может привести к более точным прогнозам.
- L1-регуляризация работает лучше, когда у модели есть несколько важных признаков и много незначительных признаков.

- L1-регуляризация может быть реализована с помощью различных методов оптимизации, включая стохастический градиентный спуск и координатный спуск.

Минусы L1-регуляризации:

- L1-регуляризация не работает хорошо, когда все признаки важны: она может уменьшить слишком много весов и потерять информацию.
- L1-регуляризация может быть менее эффективной, чем L2-регуляризация, если признаки коррелируют друг с другом, потому что она не может различать между ними.
- L1-регуляризация может быть более сложной для оптимизации, чем L2-регуляризация, потому что функция потерь не является дифференцируемой в точках, где весы равны нулю.

## L2-регуляризация

L2-регуляризация (или Ridge) - это метод регуляризации в машинном обучении, который использует квадратичную сумму весов модели, чтобы ограничить их величину. Он работает путем добавления слагаемого, пропорционального сумме квадратов весов модели, к общей функции потерь модели (Уравнение 3.10).

$$L_{L2} = L + \lambda \sum_{j=1}^d w_j^2, \quad (3.10)$$

где  $L$  - функция потерь без регуляризации,  $\lambda$  - гиперпараметр, который контролирует силу регуляризации,  $d$  - количество признаков,  $w_j$  - вес признака  $j$ .

Плюсы L2-регуляризации:

- L2-регуляризация помогает уменьшить вариацию весов модели, уменьшить переобучение и улучшить обобщающую способность модели.
- L2-регуляризация может быть лучше, чем L1-регуляризация, когда все признаки важны для модели, или когда признаки сильно коррелируют друг с другом.
- Функция потерь с L2-регуляризацией имеет гладкую форму и может быть легче оптимизирована с помощью градиентных методов.

Минусы L2-регуляризации:

- L2-регуляризация не уменьшает веса до нуля, поэтому она не может использоваться для отбора признаков.
- L2-регуляризация может быть менее эффективной, чем L1-регуляризация, когда модель имеет много незначительных признаков, которые можно исключить из модели.
- L2-регуляризация неспособна различать между сильно коррелирующими признаками, поэтому она может сохранить все такие признаки в модели, в то время как L1-регуляризация может установить некоторые из них в ноль.

## Отбор признаков с помощью L1-регуляризации

L1-регуляризация приводит к сжатию некоторых весов до нуля, что позволяет отбросить ненужные признаки и уменьшить размерность модели. Это происходит из-за свойств абсолютной функции, которая имеет резкий градиент в нуле. При увеличении коэффициента регуляризации  $\lambda$  модель становится более склонной к сокращению весов признаков до нуля.

Таким образом, L1-регуляризация является эффективным методом отбора признаков, так как она устанавливает нулевые веса для ненужных признаков, в результате чего они не влияют на предсказание модели. Однако, если все признаки являются важными для модели, L1-регуляризация может привести к потере точности, поскольку важные признаки также могут быть установлены в ноль.

Геометрическое объяснение, почему L1-регуляризация приводит к отбору признаков, связано с формой ограничений, наложенных на модель L1-регуляризацией.

Представим, что мы имеем двумерный набор данных с двумя признаками, и мы хотим создать модель регрессии для этого набора данных. Рассмотрим пространство параметров нашей модели, т.е. все возможные комбинации весов признаков. Это пространство параметров представляет собой двумерную плоскость. Если мы не используем регуляризацию, то модель может выбрать любую точку в этом пространстве параметров.

Когда мы применяем L1-регуляризацию, мы добавляем к функции потерь ограничение на сумму абсолютных значений весов модели. Функция ограничений имеет форму ромба с вершинами, соответствующими значениям, равным  $\pm\lambda$  по каждой координате.

Когда мы применяем L2-регуляризацию, мы добавляем к функции потерь ограничение на сумму квадратов значений весов модели. Функция ограничений имеет форму круга с радиусом, равным  $\lambda$ .

Оптимизация функции потерь с L1-регуляризацией эквивалентна выбору точки на плоскости, которая пересекается с ромбом. Эта точка находится на одной из вершин ромба, что соответствует некоторой комбинации признаков, для которых веса равны нулю. В ситуации же с L2-регуляризацией гораздо меньше шансов, что линия уровня коснется пространства ограничений в точке пересечения с одной из осей (Рисунок 3.9). Таким образом, при использовании L1-регуляризации приводит к отбору признаков путем установления нулевых весов для ненужных признаков.

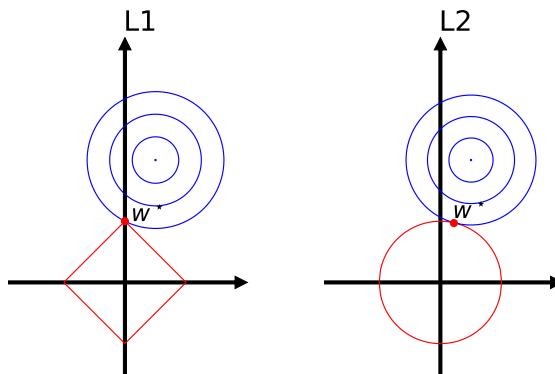


Рисунок 3.9: Отбор признаков с L1-регуляризацией

Такое геометрическое объяснение можно обобщить на пространства параметров большей размерности и на модели с несколькими признаками. Когда мы добавляем L1-регуляризацию в модель, ограничение на сумму абсолютных значений весов формирует многогранник, у которого некоторые вершины соответствуют комбинациям признаков с нулевыми весами. Оптимизация функции потерь с L1-регуляризацией приводит к выбору точки на этом многограннике, что соответствует выбору определенного подмножества признаков, для которых веса не равны нулю.

### 3.2.3 Вопросы для самопроверки

Выберите все методы регуляризации, основанные на добавлении слагаемого к функции потерь модели?

1. L1-регуляризация (Lasso)
2. L2-регуляризация (Ridge)
3. Data augmentation
4. Early stopping
5. Elastic Net
6. Dropout

Правильные ответы: 1, 2, 5

Какой из следующих признаков является признаком переобучения модели машинного обучения?

1. Низкая точность на тренировочных данных, но высокая точность на тестовых данных.
2. Высокая точность на тренировочных данных, но низкая точность на тестовых данных.
3. Равномерно высокая точность на тренировочных и тестовых данных.
4. Большое количество признаков в модели.
5. Малое количество эпох обучения модели.

Правильные ответы: 2

С помощью какого метода регуляризации можно отбирать важные признаки?

1. L1-регуляризация (Lasso)
2. L2-регуляризация (Ridge)
3. Data augmentation
4. Early stopping
5. Elastic Net
6. Dropout

Правильные ответы: 1

### 3.2.4 Резюме по разделу

Существует несколько основных методов регуляризации в машинном обучении, включая L1-регуляризацию (Lasso), L2-регуляризацию (Ridge), Elastic Net, Dropout, Data augmentation, Early stopping. Методы регуляризации, основанные на добавлении слагаемого к функции потерь модели - это L1-регуляризация (Lasso), L2-регуляризация (Ridge), Elastic Net (комбинация L1 и L2). С помощью L1 регуляризации можно отбирать важные признаки.

## 3.3 Линейная регрессия

**Цель занятия:** ученик может применить алгоритм линейной регрессии для решения задач регрессии на подготовленных и неподготовленных данных.

**План занятия:**

1. Визуальная демонстрация алгоритма
2. Описание алгоритма
3. Подготовка данных для алгоритма
4. Процесс обучения
5. Оценка качества алгоритма
6. Интерпретация признаков с помощью алгоритма
7. Модификации алгоритма
8. Область применения алгоритма
9. Плюсы и минусы алгоритма
10. Реализация алгоритма в Python

### 3.3.1 Визуальная демонстрация алгоритма

На практике достаточно часто встречаются ситуации линейной зависимости одной переменной от другой или других переменных. Линейную зависимость между двумя переменными можно вывести с помощью стандартных статистических методов, таких как корелляция. Если же целевая переменная зависит от нескольких признаков (в статистике их называют предикторами), используют более сложные модели. И первой из таких моделей является алгоритм линейной регрессии.

Визуальная демонстрация алгоритма представлена на иллюстрации 3.10.

Давайте разберемся с тем, что изображено на иллюстрации 3.10. На рисунке 3.10.1 приведена зависимость тормозного пути автомобиля от его скорости как пример построения линейной регрессии с одним признаком. Такая зависимость аппроксимируется линией и может быть визуализирована в двумерном пространстве. Алгоритм начинает поиск оптимального решения со случайных значений (синяя линия) и постепенно приближается к оптимальному значению (красная линия) (Рисунок 3.10.2). Если в наборе данных содержится два признака, то оптимальные значения будут аппроксимироваться плоскостью в трехмерном пространстве (Рисунок 3.10.3).

В линейной регрессии мы вычисляем ошибку модели как расстояние между каждой точкой данных и гиперплоскостью, которая является аппроксимацией линейной связи между независимыми и зависимой переменными. Это расстояние измеряется вдоль нормали (или перпендикуляра) к гиперплоскости.

Можно представить себе, что мы опускаем нормаль от каждой точки данных до гиперплоскости, получая расстояние между точкой данных и гиперплоскостью. Это расстояние называется остаточной суммой (residual sum) и является мерой того, насколько хорошо модель соответствует данным. В линейной регрессии мы стремимся минимизировать остаточную сумму, настраивая коэффициенты гиперплоскости таким образом, чтобы она была максимально близка к данным.

Для модели с  $d$  признаками (исключая свободный коэффициент), линейная регрессия аппроксимируется гиперплоскостью в  $d+1$  мерном пространстве. Каждый признак умножается

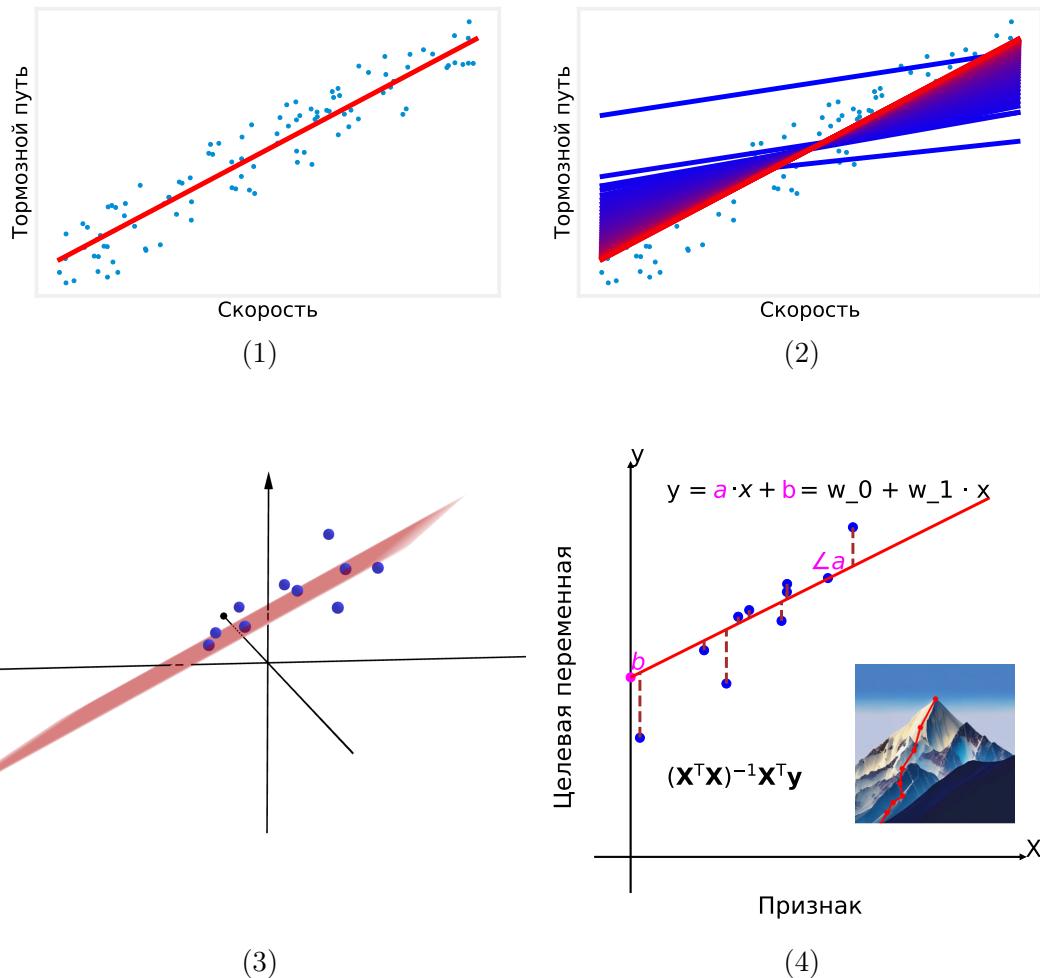


Рисунок 3.10: Визуальная демонстрация алгоритма линейной регрессии

на соответствующий вес. Также в модель добавляется настраиваемый свободный коэффициент ( $b$  или  $w_0$ ), который позволяет модели не выдавать нулевой прогноз при нулевых значениях для остальных признаков. Для свободного коэффициента обычно добавляется «фиктивный признак», все значения которого равны 1. Предсказание модели в таком случае - это скалярное произведение вектора весов на вектор признаков. Рисунок 3.10.4) демонстрирует математическую суть линейной регрессии: в двумерном случае линейная регрессия задается хорошо известным еще со школы уравнением прямой (Уравнение 3.11).

$$y = a \cdot x + b,$$

где  $y$  - целевая (зависимая) переменная,  
 $x$  - признак (независимая переменная),  
 $a$  - угол наклона прямой (slope),

$b$  - свободный коэффициент (intercept или bias), точка пересечения с осью  $y$ .

В терминах машинного обучения коэффициенты  $b$  и  $a$  обозначаются как  $w_1$  и  $w_2$  соответственно (веса модели). Для нахождения коэффициентов  $b$  и  $a$  можно пользоваться различными

методами: аналитическим (Формула  $(X^T X)^{-1} X^T y$ , не всегда применима) или градиентным спуском.

### 3.3.2 Описание алгоритма

Перейдем от визуального описания алгоритма линейной регрессии к примеру его использования и формальному определению.

Линейная регрессия с несколькими признаками, также называемая множественной линейной регрессией, является методом анализа данных, который позволяет определить линейную зависимость между целевой (зависимой) переменной и несколькими признаками (независимыми переменными).

Например, предположим, что вы хотите определить, какие факторы влияют на цену на недвижимость. В этом случае, зависимой переменной является цена на недвижимость, а независимыми переменными могут быть такие факторы, как площадь квартиры, количество комнат, расстояние до ближайшего метро, возраст здания и т.д.

Множественная линейная регрессия позволяет определить, как каждый из этих факторов влияет на цену недвижимости, и как их комбинация может объяснить изменения цен на рынке недвижимости.

Например, если мы построим модель множественной линейной регрессии для цен на недвижимость, используя площадь квартиры, количество комнат и расстояние до ближайшего метро как независимые переменные, то мы можем получить следующее уравнение:

$$\text{Цена} = 5000 + 100 * \text{Площадь} + 2000 * \text{Комнаты} - 500 * \text{Расстояние}$$

В этом уравнении коэффициент перед каждой независимой переменной (площадью, количеством комнат и расстоянием до ближайшего метро) показывает, как каждая из этих переменных влияет на цену на недвижимость. Например, увеличение площади квартиры на 1 кв. метр приведет к увеличению цены на 100 единиц, увеличение количества комнат на 1 приведет к увеличению цены на 2000 единиц, а увеличение расстояния до ближайшего метро на 1 км приведет к уменьшению цены на 500 единиц.

Таким образом, множественная линейная регрессия позволяет определить, как каждый из факторов влияет на зависимую переменную, а также как их комбинация влияет на изменения цен на рынке недвижимости.

Дадим формальное определение линейной регрессии.

Линейная регрессия - это метод машинного обучения, который используется для оценки линейной зависимости между набором признаков (независимых переменных) и целевой (зависимой) переменной. Линейная регрессия позволяет найти линейную функцию, которая наилучшим образом описывает зависимость между признаками и целевой переменной.

Математически линейная регрессия определяется следующим образом:

Пусть  $\mathbf{X}$  - матрица признаков размерности  $n \times d$ , где каждая строка представляет собой один наблюдаемый пример, а каждый столбец - один признак. Пусть  $\mathbf{y}$  - вектор целевых переменных размерности  $n \times 1$ . Тогда модель линейной регрессии выглядит следующим образом (Уравнение 3.12).

$$\hat{\mathbf{y}} = \mathbf{X}\mathbf{w} + \mathbf{b},$$

где  $\mathbf{w}$  - вектор весов размерности  $n \times 1$ ,  $\mathbf{b}$  - смещение (bias), (3.12)

$\hat{\mathbf{y}}$  - вектор предсказанных значений целевой переменной.

Если для  $\mathbf{b}$  добавить фиктивный признак, равный 1 для всех примеров, то уравнение примет еще более простой вид произведения матрицы признаков на вектор весов (Уравнение 3.13).

$$\hat{\mathbf{y}} = \mathbf{X} \cdot \mathbf{w} \quad (3.13)$$

Задача линейной регрессии заключается в нахождении оптимальных значений вектора весов  $\mathbf{w}$  и смещения  $\mathbf{b}$  таким образом, чтобы минимизировать ошибку (в примере дана среднеквадратичная ошибка) между предсказанными и реальными значениями целевой переменной (Уравнение 3.14):

$$\min_{w,b} \frac{1}{m} \sum_{i=1}^m (\hat{y}_i - y_i)^2$$

где  $\hat{y}_i$  - предсказанное значение целевой переменной для  $i$ -го примера,  $(3.14)$

$y_i$  - реальное значение целевой переменной для  $i$ -го примера,  $w$  - вектор весов,

$b$  - смещение (bias), а  $m$  - количество примеров в обучающем наборе.

### 3.3.3 Подготовка данных для алгоритма

Перед применением алгоритма линейной регрессии необходимо выполнить следующие шаги подготовки данных:

- **Очистка данных:** удаление неполных или некорректных записей, заполнение пропущенных значений и преобразование данных в формат, подходящий для анализа.
- **Нормализация данных:** приведение данных к общему масштабу, чтобы каждый признак имел одинаковый вклад в анализ. Нормирование признаков является важным шагом в предобработке данных перед использованием линейной регрессии. Это связано с тем, что линейная регрессия оценивает веса (коэффициенты) каждого признака в модели, чтобы предсказать целевую переменную. Если масштабы разных признаков значительно отличаются друг от друга, то оценка весов может быть смещена в сторону признаков с большими значениями. Это может привести к неверным выводам и плохому качеству предсказания. Обычно используются два основных способа нормирования:
  - Стандартизация: признаки масштабируются так, чтобы их среднее значение было равно 0, а стандартное отклонение было равно 1.
  - Нормализация: признаки масштабируются так, чтобы их значения находились в диапазоне от 0 до 1.

После нормирования признаков, каждый из них будет иметь примерно одинаковый вклад в предсказание целевой переменной, что позволит линейной регрессии более точно оценивать их веса и делать более точные прогнозы.

- **Разбиение данных на обучающую и тестовую выборки:** обучающая выборка используется для обучения модели, а тестовая выборка - для оценки ее точности.

### 3.3.4 Процесс обучения

Для решения этой задачи используются различные методы, такие как аналитическое решение и метод градиентного спуска. Результатом обучения модели линейной регрессии является набор оптимальных значений весов  $\mathbf{w}$  и смещения  $\mathbf{b}$ , которые могут быть использованы для предсказания целевой переменной для новых наблюдений.

## Аналитическое решение линейной регрессии

Существует аналитическое решение для линейной регрессии (Уравнение 3.15).

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

где  $X$  — матрица объекты-признаки,  $y$  - вектор ответов для объектов, (3.15)

$w^*$  — вектор оптимальных весов.

Давайте рассмотрим, как оно получено. Для вывода нормального уравнения линейной регрессии рассмотрим многомерную линейную регрессионную модель (Уравнение 3.16).

$$y_i = w_0 + w_1 x_{i1} + w_2 x_{i2} + \cdots + w_d x_{id} + \epsilon_i, i = 1, 2, \dots, n$$

где  $y_i$  - значение зависимой переменной для  $i$ -го наблюдения,

$x_{i1}, x_{i2}, \dots, x_{id}$  - значения  $d$  независимых переменных

(признаков) для  $i$ -го наблюдения,  $w_0, w_1, \dots, w_d$  - параметры регрессии,

$\epsilon_i$  - случайная ошибка или шум, которая имеет нулевое среднее и постоянную дисперсию. (3.16)

Для удобства обозначим  $X$  матрицей признаков размерности  $n \times (d+1)$ , в которой первый столбец заполнен единицами, и  $w$  вектором размерности  $(d+1) \times 1$ , состоящим из параметров регрессии:

$$X = \begin{bmatrix} 1 & w_{11} & w_{12} & \cdots & w_{1d} \\ 1 & w_{21} & w_{22} & \cdots & w_{2d} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & w_{n1} & w_{n2} & \cdots & w_{nd} \end{bmatrix}, w = \begin{bmatrix} w_1 \\ w_2 \\ \vdots \\ w_d \end{bmatrix}$$

Тогда модель может быть записана в матричной форме:

$$y = X \cdot w + \epsilon$$

Для нахождения оценок коэффициентов регрессии  $\hat{w}$ , которые минимизируют сумму квадратов ошибок, воспользуемся методом наименьших квадратов. Сначала нам нужно определить функцию ошибок  $L(w)$ , которая является суммой квадратов отклонений прогнозируемых значений от фактических значений:

$$L(w) = \sum_{i=1}^n (y_i - w_0 - w_1 x_{i1} - w_2 x_{i2} - \cdots - w_d x_{id})^2$$

Для минимизации функции  $L(w)$  найдем ее производные по каждому элементу вектора  $w$  и приравняем их к нулю:

$$\frac{\partial L(w)}{\partial w} = 2X^T(Xw - y) = 0,$$

где  $\frac{\partial L(w)}{\partial w}$  - частные производные  $L(w)$  по весам,

$X^T$  - транспонированная матрица  $X$ , а  $y$  - вектор значений целевой переменной.

Умножив обе части уравнения на  $X^T$ , получим:

$$X^T X w = X^T y$$

Это называется нормальным уравнением линейной регрессии. Решив его относительно вектора параметров  $w$ , получим:

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

Таким образом, мы получили аналитическую формулу для оценки коэффициентов линейной регрессии. Оценки параметров  $w^*$ , которые минимизируют сумму квадратов ошибок, находятся как решение нормального уравнения.

Уравнение  $w = (X^T X)^{-1} X^T y$ , которое также называется «нормальным уравнением», очень часто не применимо на практике в силу следующих причин:

- Большой размер матрицы  $X$ : на обращение матрицы требуется порядка  $n^3$  операций
- Несуществование обратной матрицы: Если матрица  $(X^T X)$  не имеет обратной матрицы, то уравнение не может быть использовано. Это может произойти, например, когда столбцы матрицы линейно зависимы.
- Некорректность модели: Если модель линейной регрессии неправильно специфицирована, то уравнение не даст правильных результатов. Например, если зависимость между переменными является нелинейной, то линейная регрессия не будет работать правильно.
- Наличие выбросов: Если в данных есть выбросы, то уравнение может дать неверные результаты. Выбросы могут сильно влиять на оценку параметров модели и приводить к неправильным выводам.
- Недостаточный размер выборки: Если выборка слишком мала, то уравнение может давать неустойчивые оценки параметров модели. В таких случаях необходимо использовать более сложные методы оценки параметров.
- Нарушение предположений модели: Если предположения модели о распределении ошибок не соблюдаются, то уравнение может давать неверные результаты. Например, если ошибки не являются нормально распределенными или имеют гетероскедастичность, то уравнение может дать неправильные оценки параметров модели.

### Линейная регрессия в матричной форме

На практике гораздо чаще применяются численные методы решения линейной регрессии, такие как градиентный спуск. Функция ошибки для линейной регрессии может быть записана так:

$$L(w) = \frac{1}{2n} * \sum_{i=1}^n (y_i - Xw)^2,$$

где  $y$  - вектор ответов,  $X$  - матрица объекты-признаки,

$w$  - вектор параметров,  $n$  - количество наблюдений.

Градиентный спуск находит минимум функции ошибки, обновляя параметры модели на каждой итерации в направлении антиградиента функции ошибки:

$$w = w - \alpha \frac{1}{n} X^T (Xw - y),$$

где  $\alpha$  - скорость обучения, которая контролирует ,

размер шага на каждой итерации.

Градиентный спуск повторяется до тех пор, пока значение функции ошибки не перестанет существенно изменяться или пока не будет достигнуто максимальное количество итераций.

Использование градиентного спуска для оптимизации линейной регрессии имеет некоторые преимущества, такие как возможность работать с большими объемами данных и возможность настройки гиперпараметров, таких как скорость обучения.

**Выходные параметры:** Оптимальные параметры  $w^*$

- 1: Задание параметров обучения: скорости обучения  $\alpha$ , числа эпох обучения  $n_{\text{epochs}}$
- 2: Инициализация весов  $\mathbf{w}$
- 3: **for**  $i = 1$  to  $n_{\text{epochs}}$  **do**
- 4:     Вычисление предсказания модели  $\hat{\mathbf{y}}_i = \mathbf{X} \cdot \mathbf{w}$
- 5:     Вычисление ошибок  $errors = \hat{\mathbf{y}}_i - \mathbf{y}$
- 6:     Вычисление градиента  $\nabla L(\mathbf{w}) = \frac{1}{n} \cdot \mathbf{X}^T \cdot errors$
- 7:     Обновление весов  $\mathbf{w} = \mathbf{w} - \alpha \nabla L(\mathbf{w})$
- 8: **end for**=0

Здесь  $n$  - это количество обучающих примеров. Метод градиентного спуска используется для обновления весов модели на каждом шаге обучения.

### 3.3.5 Оценка качества алгоритма

Для оценки качества алгоритма линейной регрессии часто используют следующие метрики:

- **Mean Squared Error (MSE):** средняя квадратичная ошибка между прогнозами и истинными значениями.
- **Mean Absolute Error (MAE):** средняя абсолютная ошибка между прогнозами и истинными значениями.
- **Huber loss:** комбинация MSE и MAE, которая более устойчива к выбросам в данных, чем MSE

### 3.3.6 Интерпретация признаков с помощью алгоритма

Интерпретация признаков в линейной регрессии заключается в понимании того, как каждый признак влияет на зависимую переменную в модели. Это может помочь понять, какие признаки наиболее важны для прогнозирования целевой переменной и как они связаны с ней.

Наиболее простым способом оценить влияние того или иного признака - посмотреть на весовой коэффициент при признаке. Коэффициенты признаков в линейной регрессии представляют величину и направление влияния каждого признака на зависимую переменную. Если значение признака большое, можно сказать, что его значение также большое (при условии нормализованных данных). Если коэффициент положительный, то увеличение значения признака будет приводить к увеличению значения зависимой переменной. Если коэффициент отрицательный, то увеличение значения признака будет приводить к уменьшению значения зависимой переменной.

### 3.3.7 Модификации алгоритма

Существует несколько модификаций алгоритма линейной регрессии, которые могут быть применены в зависимости от задачи и данных. Некоторые из них:

- **Ridge regression (Гребневая регрессия):** добавление регуляризации L2 в функцию потерь для борьбы с переобучением. Это позволяет уменьшить веса модели, чтобы она была менее склонна к переобучению и более устойчива к шуму в данных.
- **Lasso regression (Лассо регрессия):** добавление регуляризации L1 в функцию потерь, чтобы стимулировать разреженность весов модели. Это означает, что некоторые веса будут установлены на ноль, что может помочь идентифицировать наиболее важные признаки.
- **Elastic Net:** комбинация L1 и L2 регуляризации в функции потерь. Это позволяет улучшить баланс между разреженностью и устойчивостью к шуму.
- **Polynomial regression (Полиномиальная регрессия):** расширение линейной регрессии путем добавления полиномиальных признаков для учета нелинейных зависимостей между признаками и целевой переменной.
- **Robust regression (Робастная регрессия):** модификация линейной регрессии, которая более устойчива к выбросам в данных. Она использует альтернативные функции потерь, такие как Huber loss, которые уменьшают вклад выбросов в обучении модели.

### 3.3.8 Область применения алгоритма

Линейная регрессия является одним из самых простых и наиболее распространенных методов машинного обучения, и может быть применена во многих областях, включая:

- **Финансовый анализ:** линейная регрессия может использоваться для прогнозирования цен акций, доходности инвестиций и других финансовых показателей.
- **Маркетинг:** линейная регрессия может использоваться для анализа данных о продажах и прогнозирования спроса на товары и услуги.
- **Медицинская статистика:** линейная регрессия может использоваться для анализа данных о заболеваемости и смертности, прогнозирования риска развития заболеваний и определения факторов, влияющих на здоровье.
- **Инженерия:** линейная регрессия может использоваться для анализа данных в различных областях, таких как электроника, механика и транспорт, для прогнозирования характеристик и поведения систем.
- **Социальные науки:** линейная регрессия может использоваться для анализа социальных данных, таких как опросы общественного мнения, для прогнозирования выборов, определения влияния социальных факторов на поведение людей и т.д.

В целом, линейная регрессия может быть использована для анализа любых данных, для которых есть зависимость между одной или несколькими признаками и целевой переменной. Однако, следует учитывать, что линейная регрессия может быть неэффективной для данных с нелинейными зависимостями и слабыми связями между переменными.

### 3.3.9 Плюсы и минусы алгоритма

Алгоритм линейной регрессии имеет ряд преимуществ и недостатков.

#### Плюсы:

- Простота реализации и понимания. Это один из самых простых алгоритмов машинного обучения.
- Может использоваться для задач регрессии и оценки влияния признаков.

- Эффективен на больших выборках и может быстро обучаться на большом количестве признаков.
- Хорошо интерпретируется и может помочь понять важность каждого признака.

#### Минусы:

- Чувствительность к выбросам и шуму в данных.
- Требует выполнения предположения о линейной зависимости между признаками и целевой переменной, что может быть проблематично в некоторых случаях.
- Не учитывает взаимодействие между признаками, что может привести к недооценке важности некоторых признаков.
- Может быть склонен к переобучению, если в модели слишком много признаков или коэффициенты признаков сильно отличаются друг от друга.

В целом, алгоритм линейной регрессии также имеет простую и понятную логику, но его эффективность может быть низкой в случае шумных данных и если не выполняются предположения о линейной зависимости между признаками и целевой переменной.

### 3.3.10 Реализация алгоритма в Python

В библиотеке Scikit-learn в модуле linear\_model реализован класс **LinearRegression** для линейной регрессии.

Класс LinearRegression имеет следующие параметры:

- **fit\_intercept**: булевый параметр, указывающий, должно ли применяться смещение (intercept/bias). По умолчанию равен True.
- **copy\_X**: булевый параметр, указывающий, должны ли данные быть скопированы перед обучением модели. По умолчанию равен True.
- **n\_jobs**: количество параллельных задач, которые используются для расчета. По умолчанию равен None, что означает использование всех доступных процессоров.

Класс LinearRegression имеет методы **fit(X, y)** для обучения модели на данных X и y, а также метод **predict(X)** для предсказания целевых значений для новых данных X. Кроме того, класс LinearRegression имеет методы **score(X, y)** и **get\_params()** для получения коэффициента детерминации (R-квадрат) и параметров модели соответственно.

### 3.3.11 Вопросы для самопроверки

Как чаще всего происходит обучение алгоритма линейной регрессии?

1. алгоритм просто запоминает обучающую выборку, чтобы затем измерить расстояние от нового объекта до объектов в обучающей выборке
2. с помощью градиентного спуска
3. с помощью метода Ньютона

Правильные ответы: 2

Как вычисляется значение для нового объекта в задаче регрессии с помощью алгоритма линейной регрессии?

1. Значение вычисляется как среднее значение целевой переменной в обучающей выборке.
2. Значение вычисляется как скалярное произведение вектора признаков нового объекта на вектор весов модели.
3. Значение вычисляется как медианное значение целевой переменной в обучающей выборке.
4. Значение вычисляется как максимальное значение целевой переменной в обучающей выборке.

Правильные ответы: 2

Выберите плюсы алгоритма линейной регрессии:

1. Может быть склонен к переобучению, если в модели слишком много признаков или коэффициенты признаков сильно отличаются друг от друга.
2. Может использоваться для задач регрессии и оценки влияния признаков.
3. Не учитывает взаимодействие между признаками, что может привести к недооценке важности некоторых признаков.
4. Простота реализации и понимания. Это один из самых простых алгоритмов машинного обучения.
5. Требует выполнения предположения о линейной зависимости между признаками и целевой переменной, что может быть проблематично в некоторых случаях.
6. Хорошо интерпретируется и может помочь понять важность каждого признака.
7. Чувствительность к выбросам и шуму в данных.
8. Эффективен на больших выборках и может быстро обучаться на большом количестве признаков.

Правильные ответы: 2, 4, 6, 8

### 3.3.12 Резюме по разделу

**Алгоритм линейной регрессии** - это метод машинного обучения, который используется для аппроксимации линейной зависимости между независимыми и зависимыми переменными. Алгоритм предсказывает значение зависимой переменной для новых наблюдений, основываясь на линейной комбинации значений независимых переменных и коэффициентов регрессии.

**Коэффициенты регрессии** - это параметры модели, которые определяют веса независимых переменных в предсказании зависимой переменной. Чтобы определить коэффициенты, используются такие функционалы ошибок, как MSE, MAE и функционал ошибок Хьюбера.

**Основными преимуществами линейной регрессии являются простота и интерпретируемость модели**, а также возможность использования для прогнозирования непрерывных значений. Недостатками алгоритма являются чувствительность к выбросам и нелинейным зависимостям между переменными.

Также существуют модификации алгоритма, такие как регуляризованная линейная регрессия, которая позволяет уменьшить переобучение и улучшить обобщающую способность модели, а также линейная регрессия с использованием полиномиальных признаков, которая может улучшить точность модели при нелинейной зависимости между переменными.

## 3.4 Резюме по модулю

В этом модуле мы рассмотрели такие темы, как градиентный спуск для обучения дифференцируемых моделей машинного обучения, регуляризация моделей машинного обучения для борьбы с переобучением, а также познакомились с алгоритмом линейной регрессии.

# Глава 4

## Линейная классификация. Логистическая регрессия. Метод опорных векторов.

### 4.1 Линейная классификация

Ранее мы рассматривали задачу регрессии. Теперь давайте поговорим о задаче классификации. Для начала проведем сравнение этих подходов.

#### 4.1.1 Сравнение задач регрессии и классификации

Задачи регрессии и классификации отличаются по своей природе, целям и используемым методам.

Задача регрессии заключается в предсказании непрерывного значения целевой переменной на основе входных признаков. То есть, мы строим модель, которая может предсказывать, например, цену на недвижимость, уровень дохода или количество продаж в зависимости от различных факторов. Методы, используемые для решения задач регрессии, включают метод k ближайших соседей, линейную регрессию, решающие деревья, случайный лес, градиентный бустинг и нейронные сети.

Задача классификации, с другой стороны, заключается в предсказании дискретных значений целевой переменной или вероятностей классов на основе входных признаков. То есть, мы строим модель, которая может предсказывать, например, является ли электронное письмо спамом или не спамом, какой класс имеет изображение, какое слово было произнесено на записи и т.д. Методы, используемые для решения задач классификации, включают метод k ближайших соседей, логистическую регрессию, метод опорных векторов, решающие деревья, случайный лес, градиентный бустинг и нейронные сети.

Главным отличием между задачами регрессии и классификации является тип целевой переменной: непрерывный для регрессии и дискретный для классификации. Это приводит к различным метрикам оценки модели и методам её построения. Например, для задач регрессии обычно используются метрики, такие как средняя абсолютная ошибка (MAE), среднеквадратическая ошибка (MSE), а для задач классификации — точность, полнота, F1-мера и др.

### 4.1.2 Трансформация линейной регрессии в линейную классификацию

Главной проблемой перенастройки модели линейной регрессии на задачу классификации является то, что линейная регрессия выдает непрерывное вещественное число, в то время как классификация осуществляется для дискретного набора классов.

Для преобразования модели линейной регрессии в модель линейной классификации можно использовать функцию активации, которая преобразует непрерывный выход модели в дискретное значение класса или вероятность класса. Среди таких функций — сигмоидальная функция и функция softmax. Мы изучим их далее в этом модуле. В простейшем случае модель классификации возвращает только два значения, которые обычно называют положительным и отрицательным классом. Такую классификацию называют бинарной. Давайте разберемся, как она устроена.

### 4.1.3 Бинарная классификация

Бинарная классификация - это задача машинного обучения, в которой требуется отнести объекты к одному из двух классов на основе набора признаков. Другими словами, бинарная классификация решает задачу разделения объектов на две группы, где каждая группа соответствует одному из двух классов.

Примерами бинарной классификации могут служить определение, является ли электронное письмо спамом или не спамом, прогнозирование, выздоровеет ли пациент после операции или нет, определение, является ли транзакция мошеннической или нет и т.д.

Одной из наиболее распространенных функций активации для бинарной классификации является функция порога, которая возвращает 1, если выход модели больше определенного порога (threshold), и 0 или -1 в противном случае. Это можно выразить следующей формулой (Формула 4.1):

$$y = \begin{cases} 1 & \text{если } w \cdot x + b \geq \text{threshold} \\ 0 \text{ или } -1 & \text{иначе} \end{cases} \quad (4.1)$$

Мы для удобства будем пользоваться метками классов «+1» и «-1». В таком случае, для линейной классификации модель по сути предсказывает знак скалярного произведения вектора весов на вектор признаков объекта.

### Геометрическая интерпретация бинарной классификации

Давайте визуализируем разделение классов «+1» и «-1» с помощью гиперплоскости. Гиперплоскость - это n-мерное обобщение понятия прямой или плоскости в пространстве большей ( $n+1$ ) размерности. Она определяется уравнением, которое состоит из суммы произведений коэффициентов на переменные, равной некоторой константе. (Рисунок 4.1). Для получения разделяющей гиперплоскости в задаче линейной классификации можно использовать вектор нормали. Этот вектор задает гиперплоскость, перпендикулярен ей и определяется коэффициентами модели линейной классификации. Вектор нормали направлен в сторону положительного класса. В общем случае вектор нормали можно задать по формуле 4.2.

$$\mathbf{w} = (w_1, w_2, \dots, w_d) \quad (4.2)$$

где  $d$  - число признаков,  $w_1, w_2, \dots, w_d$  - коэффициенты модели.

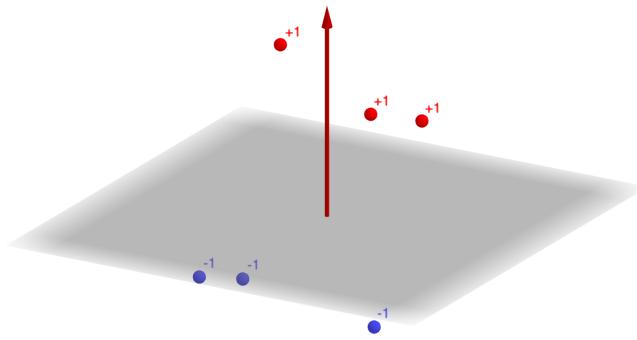


Рисунок 4.1: Классификация с помощью гиперплоскости

Для получения единичного вектора нормали  $\mathbf{w}$  его необходимо нормировать на его длину (Формула 4.3).

$$\mathbf{w}_{\text{norm}} = \frac{\mathbf{w}}{\|\mathbf{w}\|} \quad (4.3)$$

Таким образом, если мы знаем коэффициенты модели линейной классификации, то мы можем легко получить вектор нормали, который можно использовать для построения разделяющей гиперплоскости. Если скалярное произведение вектора признаков объекта на вектор весов модели  $x_i \cdot w \geq 0$ , модель отнесет объект к положительному классу, иначе — к отрицательному

В случае двух признаков модель будет использовать линию в качестве разделяющей классифицирующей поверхности (Рисунок 4.2).

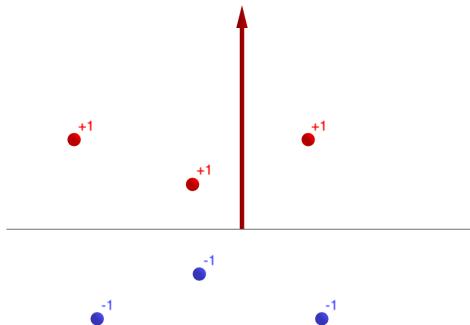


Рисунок 4.2: Линия в качестве разделяющей классифицирующей поверхности

Подытоживая вышесказанное, можно сказать, что простейший бинарный классификатор можно построить с помощью пороговой функции, учитывая только знак скалярного произведения вектора признаков объекта на вектор весов объекта.

## Отступ модели на объекте

На рисунке 4.2 мы можем заметить, что полезно учитывать не только знак скалярного произведения, но и расстояние объекта от разделяющей гиперплоскости. Для учета этого расстояния существует мера, называемая отступом.

Отступ (англ. margin) в линейной классификации - это мера того, насколько «уверена» модель в своем прогнозе для конкретного объекта. Он определяется как произведение предсказанной моделью метки класса на расстояние от объекта до гиперплоскости (или прямой в случае двумерной линейной классификации), разделяющей классы.

Формально, для объекта с признаками  $x$  и меткой  $y$  отступ можно выразить следующим образом (Формула 4.4).

$$M(x_i, y_i) = y_i * x_i \cdot w$$

где  $x_i \cdot w$  — скалярное произведение вектора признаков объекта на вектор весов, (4.4)

$y_i$  — метка класса (-1 или +1).

Таким образом, если модель правильно предсказала метку класса для объекта и расстояние от него до гиперплоскости большое (то есть объект находится далеко от гиперплоскости), то отступ будет большим и модель будет «уверена» в своем предсказании. Если же объект находится близко к гиперплоскости, то отступ будет маленьким и модель будет менее «уверена» в своем предсказании. Если модель неправильно предсказала метку класса для объекта и отступ большой, возможно, модель работает неправильно или данный объект является выбросом (Рисунок 4.3).

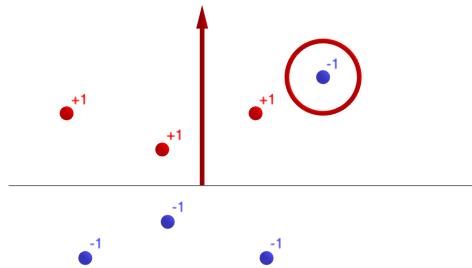


Рисунок 4.3: Выброс с большим отступом

## Функции потерь в бинарной классификации

Функции потерь (или функции ошибки) в задаче бинарной классификации используются для измерения расхождения между предсказанными и реальными метками классов. Они применяются в процессе оптимизации модели, когда мы пытаемся найти такие значения параметров модели, которые минимизируют ошибку.

Рассмотрим некоторые функции потерь в бинарной классификации.

Бинарная функция потерь с индикатором определяется следующим образом (Формула 4.5).

$$L(y, \hat{y}) = \mathbb{1}(y \neq \hat{y}),$$

где  $y$  - истинный класс объекта (-1 или 1),  
 $\hat{y}$  - предсказанный класс объекта (-1 или 1),  
 $\mathbb{1}$  - индикаторная функция, которая равна 1, если ее аргумент истинен,  
и 0 в противном случае.

(4.5)

Функция потерь равна 1, если предсказанный класс  $\hat{y}$  не соответствует истинному классу  $y$ , и равна 0 в противном случае. Минимизация бинарной функции потерь с индикатором в задаче бинарной классификации является задачей оптимизации. Однако, такая функция потерь обычно не используется, потому что она не дифференцируема и не может быть использована в градиентных методах оптимизации, которые требуют вычисления градиента функции потерь по параметрам модели.

Для того, чтобы получить дифференцируемую функцию потерь, бинарная функция потерь с индикатором должна быть преобразована. Можно в индикаторную функцию потерь в качестве аргумента передавать отступ на объекте, сравнивая его с нулем (Формула 4.6).

$$L(M(x, y)) = \mathbb{1}(M(x, y) < 0),$$

где  $M(x, y)$  - отступ на объекте,  
 $\mathbb{1}$  - индикаторная функция, которая равна 1, если ее аргумент истинен,  
и 0 в противном случае.

(4.6)

В этом случае получается пороговая функция потерь. Такая функция по-прежнему недифференцируема. Нам необходимо сделать еще один шаг. Можно добавить некоторую другую дифференцируемую функцию потерь  $\tilde{L}(M(x, y))$ , которая будет верхней оценкой для  $L(M(x, y))$  с расчетом на то, что оптимизация  $\tilde{L}(M(x, y))$  приведет также к оптимизации  $L(M(x, y))$ . Для  $\tilde{L}(M(x, y))$  применяются различные функции потерь:

- Логистическая функция потерь. Логистическая функция потерь используется в задачах классификации для оценки ошибки модели. Функция определяется следующим образом: (Формула 4.7).

$$L(M) = \log(1 + e^{-M}),$$

где  $M$  - это отступ на объекте ( $y\hat{y}$ ).

(4.7)

Функция потерь принимает на вход отрицательные значения  $M$ , и возрастает монотонно к 0 при  $M \rightarrow -\infty$ , и к  $\log(2)$  при  $M \rightarrow +\infty$ . Это означает, что чем более уверенной является модель в отнесении объекта к одному из классов, тем меньше будет значение функции потерь.

- Функция потерь Хинджа. Для задач классификации, где  $y \in \{-1, 1\}$  - истинная метка класса, а  $\hat{y}$  - предсказанная метка класса, функция потерь Хинджа может быть выражена следующим образом: (Формула 4.8).

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

где  $\max$  - функция, возвращающая максимальное значение из двух аргументов, (4.8)  
 $M$  - это отступ на объекте ( $y\hat{y}$ ).

Эта функция возвращает 0, если предсказание верно ( $y\hat{y} \geq 1$ ), иначе возвращает значение, пропорциональное расстоянию между предсказанием и истинной меткой класса. Чем больше расстояние между ними, тем больше значение функции потерь. Можно заметить, что  $y\hat{y}$  является отступом на объекте ( $M$ ).

- Логарифмическая (Бинарная кросс-энтропия). Логарифмическая функция потерь и бинарная кросс-энтропия имеют одинаковую математическую формулу, но разные интерпретации. Различие в трактовке этих понятий заключается в разных контекстах. Логарифмическая функция потерь измеряет расхождение между фактическим распределением классов и распределением, предсказанным моделью. Бинарная кросс-энтропия измеряет расхождение между фактическим и предсказанным распределениями вероятностей для двух классов. Название «кросс-энтропия» происходит от того, что эта функция потерь измеряет расхождение между двумя вероятностными распределениями. «Кросс» здесь означает «перекрестный», так как мы сравниваем два распределения. «Бинарная» здесь означает, что мы сравниваем распределения для двух классов.
- Для двух бинарных векторов  $y$  и  $\hat{y}$ , бинарная кросс-энтропия может быть выражена следующим образом (Формула 4.9):

$$L(y, \hat{y}) = -\frac{1}{N} \sum_{i=1}^d [y_i \log \hat{y}_i + (1 - y_i) \log(1 - \hat{y}_i)] \quad (4.9)$$

где  $d$  - количество элементов в векторах  $y$  и  $\hat{y}$ ,

$y_i$  и  $\hat{y}_i$  - элементы векторов  $y$  и  $\hat{y}$  соответственно.

Графики пороговой и логистической функции, а также функции потерь Хинджа и логарифмической функции потерь (бинарной кросс-энтропии) приведены на рисунке (Рисунок 4.4).

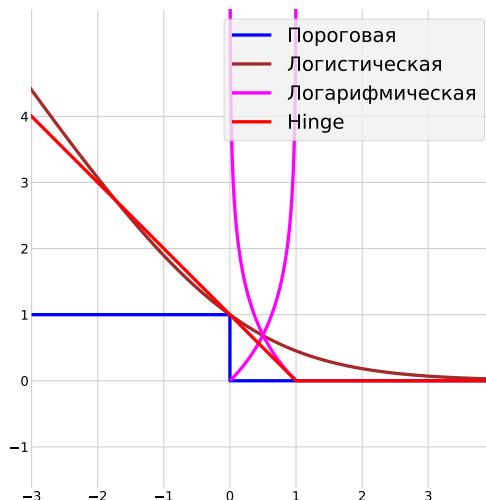


Рисунок 4.4:  $\tilde{L}(M(x, y))$  функции потерь

#### 4.1.4 Многоклассовая классификация

Также давайте рассмотрим задачи многоклассовой классификации, когда классов больше, чем два. Многоклассовая классификация - это задача классификации, где необходимо отнести объекты к одному из нескольких классов. Например, можно классифицировать изображения по видам животных (собаки, кошки, птицы и т.д.), рукописные цифры (от 0 до 9) или типы цветов.

Существует несколько подходов к решению задачи многоклассовой классификации, включая:

- Один против всех (One-vs-All) - обучение отдельной модели для каждого класса, где каждая модель разделяет этот класс от остальных. При классификации нового объекта используются все модели, и объект относится к классу, для которого модель выдала наивысшую вероятность.
- Один против другого (One-vs-One) - обучение модели для каждой пары классов, где каждая модель разделяет два класса. При классификации нового объекта каждая модель голосует за класс, к которому относится объект, и объект относится к классу, который набрал наибольшее количество голосов.
- Многоклассовая логистическая регрессия (Multinomial Logistic Regression) - обучение единственной модели, которая предсказывает вероятности принадлежности объекта к каждому классу. Для обучения модели используется перекрестная энтропия (cross-entropy) — функция потерь, которая минимизирует ошибку классификации.

Рассмотрим эти подходы подробнее.

**Один против всех (One-vs-All)** - это подход к решению задачи многоклассовой классификации, который заключается в обучении отдельной модели для каждого класса, где каждая модель разделяет этот класс от остальных (Рисунок 4.5).

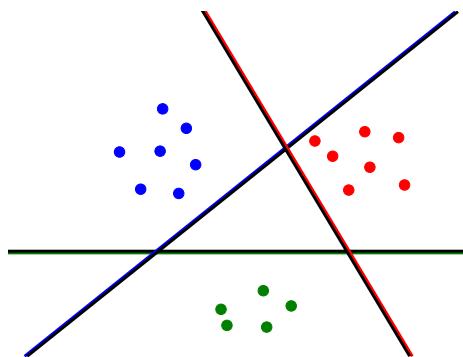


Рисунок 4.5: Подход Один против всех (One-vs-All)

Например, представим, что мы решаем задачу классификации изображений на 3 класса: кошки, собаки и птицы. В методе один против всех мы будем обучать 3 модели - для каждого класса отдельно. То есть, модель для кошек будет учиться разделять кошек от всех других животных, модель для собак - разделять собак от всех других животных, и модель для птиц - разделять птиц от всех других животных.

При классификации нового изображения мы будем применять все 3 модели и выбирать тот класс, для которого модель выдала наивысшую вероятность. Например, если модель для

кошек выдала вероятность 0.6, а модель для собак - 0.3 и для птиц - 0.1, то мы отнесем изображение к классу кошек.

Этот подход удобен, так как мы можем использовать любой классификатор, который может работать с двумя классами. Также он может быть эффективным в случаях, когда классов много, а обучающие выборки малы. Однако он может столкнуться с проблемой, если классы имеют пересекающиеся области, так как каждая модель учится только разделять свой класс от остальных, и не учитывает отношения между остальными классами.

В библиотеке scikit-learn реализован метод One-vs-All (Один против всех) для решения задач многоклассовой классификации. Он используется в моделях, которые не поддерживают прямую многоклассовую классификацию, например, в методе логистической регрессии.

В scikit-learn реализация One-vs-All выполняется автоматически при использовании метода логистической регрессии с параметром `multi_class='ovr'` (один против всех) или при использовании метода Support Vector Machine (SVM) с параметром `decision_function_shape='ovr'`.

**Один против другого (One-vs-One)** - это подход к решению задачи многоклассовой классификации, который заключается в обучении модели для каждой пары классов, где каждая модель разделяет два класса (Рисунок 4.6).

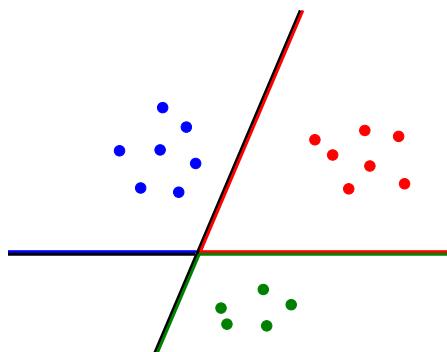


Рисунок 4.6: Подход Один против другого (One-vs-One)

Например, представим, что мы решаем задачу классификации изображений на 3 класса: кошки, собаки и птицы. В методе один против другого мы будем обучать 3 модели - одну для каждой пары классов. То есть, модель для кошек и собак будет учиться разделять кошек от собак, модель для кошек и птиц - разделять кошек от птиц, и модель для собак и птиц - разделять собак от птиц.

При классификации нового изображения мы будем применять все 3 модели и выбирать тот класс, к которому относится изображение, с учетом голосования моделей. Например, если изображение было отнесено к классу кошек 2 раза (модель для кошек и собак и модель для кошек и птиц), а к классу собак - 1 раз (модель для кошек и собак), то мы отнесем изображение к классу кошек.

Этот подход имеет преимущество в том, что он более точен, чем подход один против всех, потому что каждая модель учитывает отношения только между двумя классами. Однако, для классификации изображения требуется выполнение большего количества вычислений, так как нам нужно обучать больше моделей и выполнять больше предсказаний.

В библиотеке scikit-learn метод One-vs-One также реализован для решения задач многоклассовой классификации. В отличие от метода One-vs-All, в котором обучается один классификатор для каждого класса, метод One-vs-One обучает классификатор для каждой пары классов. Для использования метода One-vs-One в scikit-learn можно использовать классификаторы, которые поддерживают прямую многоклассовую классификацию, например, методы Support Vector Machine (SVM, `decision_function_shape='ovo'`) и Random Forest (`multi_class='ovo'`).

**Многоклассовая логистическая регрессия (Multinomial Logistic Regression)** - это метод многоклассовой классификации, который является обобщением бинарной логистической регрессии на случай, когда число классов больше двух.

В этом методе мы моделируем вероятность принадлежности объекта к каждому классу при помощи функции softmax. Функция softmax (софтмакс) - это функция, которая преобразует вектор произвольных действительных чисел в вектор вероятностей, где сумма всех вероятностей равна 1. Эта функция широко используется в машинном обучении, особенно в задачах многоклассовой классификации, где требуется присвоить каждому объекту один из нескольких классов.

Формула для функции softmax (Формула 4.10):

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^n e^{z_j}}, \quad (4.10)$$

где  $z_i$  - элемент входного вектора  $z$ , а  $n$  - количество элементов вектора  $z$ .

Таким образом, для каждого элемента  $z_i$  входного вектора  $z$ , функция softmax вычисляет экспоненту этого элемента, а затем нормирует все экспоненты путем их деления на сумму экспонент всех элементов вектора  $z$ . Результатом является вектор вероятностей, где каждый элемент соответствует вероятности принадлежности объекта к соответствующему классу.

Пример использования функции softmax: предположим, что у нас есть модель, которая должна классифицировать изображения на три класса: кошки, собаки и птицы. Модель возвращает три числа, каждое из которых соответствует вероятности принадлежности изображения к соответствующему классу. Затем мы применяем функцию softmax к этому вектору, чтобы получить вероятности для каждого класса.

Например, если модель возвращает вектор  $[1.5, 2.3, 0.8]$ , то после применения функции softmax мы получим вектор вероятностей  $[0.217, 0.536, 0.247]$ . Это означает, что модель считает, что изображение наиболее вероятно принадлежит классу собак (Рисунок 4.7).

Модель многоклассовой логистической регрессии определяется набором параметров, включающих в себя веса признаков и векторы смещения для каждого класса. Обучение модели заключается в нахождении наилучших параметров, минимизируя функцию потерь, такую как кросс-энтропийная функция потерь (cross-entropy loss). Если у нас есть  $C$  классов, и  $y_i$  обозначает истинную метку класса для  $i$ -го примера в обучающем наборе, а  $p_{i,j}$  обозначает предсказанную вероятность, что  $i$ -й пример принадлежит к классу  $j$ . Тогда кросс-энтропийная функция потерь вычисляется следующим образом (Формула 4.11):

$$L(w) = -\frac{1}{n} \sum_{i=1}^n \sum_{j=1}^C y_{i,j} \log(p_{i,j}), \quad (4.11)$$

где  $w$  - это параметры модели,  $n$  - размер обучающего набора.

Каждое слагаемое в сумме соответствует ошибке для одного примера и одного класса, и общая функция потерь является средним значением всех таких ошибок.

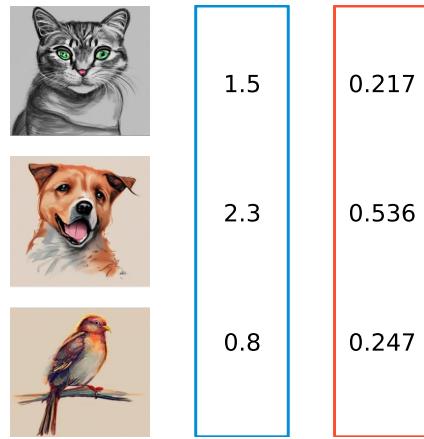


Рисунок 4.7: Демонстрация работы softmax

В кросс-энтропийной функции потерь для многоклассовой классификации мы используем метки классов в форме one-hot encoding. То есть, если пример относится к классу  $k$ , то  $y_{i,k} = 1$ , а для всех остальных классов  $y_{i,j} = 0$ . Функция потерь штрафует модель за ошибки в предсказании вероятностей для всех классов, а не только для истинного класса. Она пытается минимизировать расстояние между предсказанными вероятностями и one-hot кодировкой истинных меток классов.

После обучения модели мы можем классифицировать новые объекты, вычислив вероятности для каждого класса при помощи функции softmax и выбрав класс с наибольшей вероятностью.

По умолчанию в scikit-learn для задач многоклассовой классификации используется логистическая регрессия (LogisticRegression), которая использует softmax в качестве функции активации для расчета вероятности принадлежности каждого класса.

#### 4.1.5 Вопросы для самопроверки

Для преобразования модели линейной регрессии в модель линейной классификации используется функция активации, которая преобразует непрерывный выход модели в дискретное значение класса или вероятность класса. Какая функция активации может использоваться для этой цели?

1. MSE
2. Сигмоидальная функция
3. Функция softmax
4. Функция tanh

Правильные ответы: 2, 3

В машинном обучении отступ (margin) является расстоянием от гиперплоскости до обучающего примера. Какой из следующих вариантов ответа лучше всего описывает важность отступа в задаче классификации?

1. Отступ не является важным для задачи классификации.
2. Чем больше отступ и знак отступа правильный, тем выше вероятность правильной классификации.
3. Чем меньше отступ и знак отступа правильный, тем выше вероятность правильной классификации.
4. Отступ не имеет никакого отношения к вероятности правильной классификации.

Правильные ответы: 2

Какие подходы применяют для реализации возможности многоклассовой классификации?

1. Градиентный спуск
2. One-vs-One
3. One-vs-All
4. Multinomial Logistic Regression

Правильные ответы: 2, 3, 4

#### 4.1.6 Резюме по разделу

В задаче классификации необходимо предсказать дискретное значение класса или вероятность класса.

Бинарная классификация - это задача машинного обучения, в которой требуется отнести объекты к одному из двух классов на основе набора признаков.

Существует несколько подходов для реализации возможности многоклассовой классификации:

1. One-vs-One
2. One-vs-All
3. Multinomial Logistic Regression

## 4.2 Логистическая регрессия

**Цель занятия:** ученик может применить алгоритм логистической регрессии для решения задач классификации на подготовленных и неподготовленных данных.

**План занятия:**

- Визуальная демонстрация алгоритма
- Описание алгоритма
- Подготовка данных для алгоритма
- Процесс обучения
- Оценка качества алгоритма
- Интерпретация признаков с помощью алгоритма
- Модификации алгоритма
- Область применения алгоритма
- Плюсы и минусы алгоритма
- Реализация алгоритма в Python

### 4.2.1 Визуальная демонстрация алгоритма

Логистическая регрессия - это метод машинного обучения, который используется для решения задач бинарной классификации, т.е. для разделения данных на два класса.

Классический пример задачи, который может быть решен с помощью логистической регрессии, это предсказание вероятности того, что клиент купит продукт, основываясь на истории его покупок и других данных.

Логистическая регрессия использует логистическую функцию (сигмоиду) для прогнозирования вероятности принадлежности объекта к одному из двух классов. Входные данные, обычно представленные вектором признаков, умножаются на веса, которые настраиваются во время обучения, и затем подаются на вход логистической функции. Результатом является вероятность принадлежности объекта к положительному классу. Визуальная демонстрация алгоритма представлена на иллюстрации 4.8.

Давайте разберемся с тем, что изображено на иллюстрации 4.8. На рисунке 4.8.1 приведена зависимость целевой переменной от признака как пример построения линейной регрессии с одним признаком. Такая зависимость аппроксимируется (приближается) линией. Далее мы вводим сигмоидальную активацию ( $\sigma(x) = \frac{1}{1+e^{-x \cdot w}}$ ), чтобы сжать получившиеся значения в диапазон (0, 1) (Рисунок 4.8.2). Соответственно, применение сигмоиды к линейной комбинации весов и признаков позволяет спроектировать данные на сигмоиду (Рисунок 4.8.3) и сделать вывод о том, к какому классу принадлежит объект (Рисунок 4.8.4). В данном случае порог классификации для разделения классов установлен на уровне 0.5.

### 4.2.2 Описание алгоритма

Перейдем от визуального описания алгоритма логистической регрессии к формальному определению.

Логистическая регрессия - это метод машинного обучения, который используется для классификации данных. Он применяется для прогнозирования вероятности отнесения наблюдаемого объекта к определенному классу. Этот метод широко применяется в задачах бинарной классификации, когда требуется определить, принадлежит ли объект к одному из двух классов.

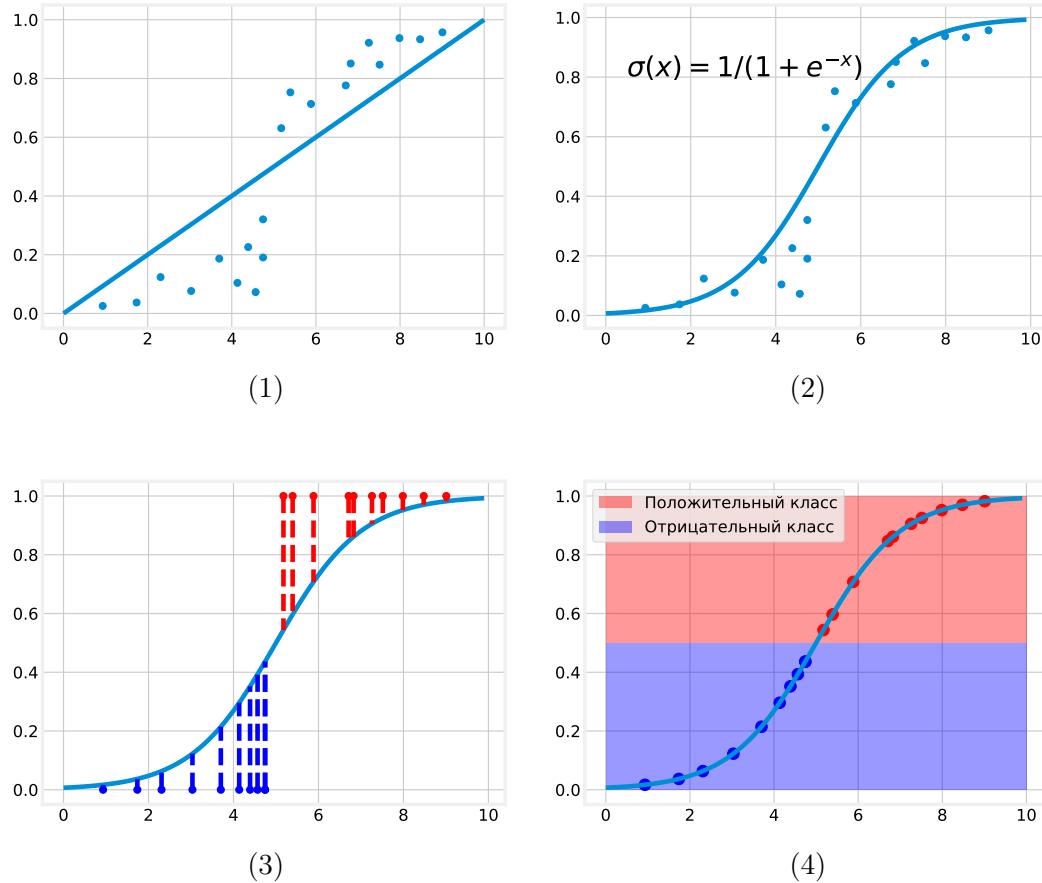


Рисунок 4.8: Визуальная демонстрация алгоритма логистической регрессии

Формальное определение логистической регрессии:

Пусть даны обучающие данные  $(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)$ , где  $x_i$  - это вектор признаков, а  $y_i$  - это метка класса для  $i$ -го объекта. Логистическая регрессия строит модель, которая связывает вектор признаков  $x$  с вероятностью  $p(y = 1|x)$  принадлежности объекта  $x$  к классу  $y = 1$ . Функция активации, используемая в логистической регрессии, называется сигмоидной функцией и определяется следующим образом (Формула 4.12).

$$p(y = 1) = \sigma(w \cdot x_i) = \frac{1}{1 + e^{-w \cdot x_i}} \quad (4.12)$$

где  $w$  - вектор весов, а  $\sigma$  - сигмоидная функция.

В качестве функции потерь в логистической регрессии обычно используется логистическая функция потерь (Формула 4.7).

Цель обучения логистической регрессии - найти вектор весов  $w$ , которые минимизируют функцию потерь на обучающих данных. Эту задачу можно решать различными численными методами оптимизации, такими как градиентный спуск.

### 4.2.3 Подготовка данных для алгоритма

Перед применением алгоритма логистической регрессии необходимо выполнить следующие шаги подготовки данных:

- **Очистка данных:** удаление неполных или некорректных записей, заполнение пропущенных значений и преобразование данных в формат, подходящий для анализа.
- **Нормализация данных:** приведение данных к общему масштабу, чтобы каждый признак имел одинаковый вклад в анализ. Нормирование признаков является важным шагом в предобработке данных перед использованием логистической регрессии. Это связано с тем, что логистическая регрессия оценивает веса (коэффициенты) каждого признака в модели, чтобы предсказать целевую переменную. Если масштабы разных признаков значительно отличаются друг от друга, то оценка весов может быть смещена в сторону признаков с большими значениями. Это может привести к неверным выводам и плохому качеству предсказания. Обычно используются два основных способа нормирования:
  - Стандартизация: признаки масштабируются так, чтобы их среднее значение было равно 0, а стандартное отклонение было равно 1.
  - Нормализация: признаки масштабируются так, чтобы их значения находились в диапазоне от 0 до 1.

После нормирования признаков, каждый из них будет иметь примерно одинаковый вклад в предсказание целевой переменной, что позволит логистической регрессии более точно оценивать их веса и делать более точные прогнозы.

- **Разбиение данных на обучающую и тестовую выборки:** обучающая выборка используется для обучения модели, а тестовая выборка - для оценки ее точности.

### 4.2.4 Процесс обучения

Процесс поиска оптимума весов в логистической регрессии градиентным спуском аналогичен тому, как это происходит в линейной регрессии, за исключением функции потерь. Алгоритм логистической регрессии получил свое название от функции потерь, с помощью которой он обучается — логистической регрессии (Формула 4.7). Давайте поймем, почему выбрана именно такая функция потерь.

Логистическая регрессия нацелена на то, чтобы предсказывать вероятность положительного класса в случае бинарной классификации. Что значит предсказывать вероятность в контексте классификации? Если модель предсказывает вероятность определенного класса, равную, допустим 60%, для определенной подвыборки, например, из 1000 объектов, то количество объектов этого класса в подвыборке должно составлять примерно 600 объектов. Тогда мы можем говорить, что модель корректно предсказывает вероятности.

Для того, чтобы получать вероятности, в логистической регрессии используется сигмода ( $\sigma(x) = \frac{1}{1+e^{-x \cdot w}}$ ), которая переводит скалярное произведение весов модели на признаки объекта в отрезок (0, 1). Если параметры модели будут подобраны правильно, мы сможем получать:

- вероятность, близкую к 1, для объектов положительного класса
- вероятность, близкую к 0, для объектов отрицательного класса.

В таком случае:

- для объектов положительного класса должно выполняться:  $\sigma(x \cdot w) \rightarrow 1, x \cdot w \rightarrow +\infty$  (сигмоида скалярного произведения стремится к 1 при стремлении скалярного произведения к  $+\infty$ )
- для объектов отрицательного класса должно выполняться:  $\sigma(x \cdot w) \rightarrow 0, x \cdot w \rightarrow -\infty$  (сигмоида скалярного произведения стремится к 0 при стремлении скалярного произведения к  $-\infty$ )

Такие условия аналогичны максимизации правильных отступов на объектах. Их можно записать с помощью такого функционала ошибки:

$$\frac{1}{n} \sum_{i=1}^n [y_i = 1] \sigma(w \cdot x_i) + [y_i = -1] (1 - \sigma(w \cdot x_i)) \rightarrow \min_w$$

У этого функционала есть существенный недостаток: он не сильно штрафует модель в случае серьезной ошибки. Если на положительном объекте модель уверена, что это отрицательный объект и сигмоида выдает значение, равное нолю, то штраф будет равен только единице. Если бы вероятность на этом объекте была равна единице, то функция потерь на этом объекте была бы минимальной и равной -1. В этом случае первое слагаемое в функции потерь было бы активировано, и мы бы взяли сигмоиду с минусом и значением -1. Однако в нашем случае сигмоида равна нулю, поэтому функция потерь также равна нулю. Соответственно, штраф равен единице (разность между 0 и -1).

Для ужесточения штрафа в формулу функционала вводят логарифм, получая логарифмический функционал:

$$\frac{1}{n} \sum_{i=1}^n [y_i = 1] \log(\sigma(w \cdot x_i)) + [y_i = -1] (1 - \log(\sigma(w \cdot x_i))) \rightarrow \min_w$$

В этом случае мы получим логарифм нуля, если модель крайне уверена в неправильном ответе. Логарифм нуля равен  $-\infty$  и, умноженный на -, даст в итоге  $+\infty$ , ужесточая таким образом штраф за уверенность в неправильном ответе.

После введения логарифма в функцию потерь над логарифмической функцией производят ряд предобразований, получая в итоге логистическую функцию потерь:

$$L(M) = \log(1 + e^{-M})$$

Далее стандартным алгоритмом градиентного спуска находят минимум этого функционала ошибки.

#### 4.2.5 Оценка качества алгоритма

Для оценки качества алгоритма логистической регрессии часто используют следующие метрики:

- **accuracy:**  $\frac{TP+TN}{TP+TN+FP+FN}$
- **precision:**  $\frac{TP}{TP+FP}$
- **recall:**  $\frac{TP}{TP+FN}$
- **F1:**  $2 * \frac{\text{precision} * \text{recall}}{\text{precision} + \text{recall}}$

### 4.2.6 Интерпретация признаков с помощью алгоритма

Интерпретация признаков в логистической регрессии заключается в понимании того, как каждый признак влияет на зависимую переменную в модели. Это может помочь понять, какие признаки наиболее важны для прогнозирования целевой переменной и как они связаны с ней.

Наиболее простым способом оценить влияние того или иного признака - посмотреть на весовой коэффициент при признаке. Коэффициенты признаков в логистической регрессии представляют величину и направление влияния каждого признака на зависимую переменную. Если значение признака большое, можно сказать, что его значение также большое (при условии нормализованных данных). Если коэффициент положительный, то увеличение значения признака будет приводить к увеличению значения зависимой переменной. Если коэффициент отрицательный, то увеличение значения признака будет приводить к уменьшению значения зависимой переменной.

### 4.2.7 Модификации алгоритма

Существует несколько модификаций алгоритма логистической регрессии, которые могут быть применены в зависимости от задачи и данных. Некоторые из них:

- **добавление регуляризации L2:** добавление регуляризации L2 в функцию потерь для борьбы с переобучением. Это позволяет уменьшить веса модели, чтобы она была менее склонна к переобучению и более устойчива к шуму в данных.
- **добавление регуляризации L1:** добавление регуляризации L1 в функцию потерь, чтобы стимулировать разреженность весов модели. Это означает, что некоторые веса будут установлены на ноль, что может помочь идентифицировать наиболее важные признаки.
- **Мультиклассовая логистическая регрессия:** алгоритм логистической регрессии может быть расширен на случай многоклассовой классификации с помощью подходов One-vs-All, One-vs-One, softmax + cross-entropy

### 4.2.8 Область применения алгоритма

Алгоритм логистической регрессии широко используется в различных областях:

- Медицина: например, для определения вероятности заболевания у пациента на основе их медицинских данных.
- Финансы: например, для определения вероятности дефолта заемщика на основе его кредитной истории.
- Маркетинг: например, для прогнозирования вероятности покупки товара на основе поведенческих данных покупателя.
- Обработка естественного языка: например, для классификации текстовых документов или определения тональности текста.
- Компьютерное зрение: например, для классификации изображений.
- Рекомендательные системы: например, для предсказания рейтинга товара на основе истории покупок и поведения пользователя.
- Информационная безопасность: например, для определения вероятности атаки на систему на основе ее характеристик.

- Биоинформатика: например, для анализа генетических данных и определения вероятности развития заболевания.

В целом, алгоритм логистической регрессии может быть применен в любой области, где необходимо решать задачи бинарной или многоклассовой классификации на основе набора признаков.

#### 4.2.9 Плюсы и минусы алгоритма

Алгоритм логистической регрессии имеет ряд преимуществ и недостатков.

##### Плюсы:

- Простота реализации и понимания. Это один из самых простых алгоритмов машинного обучения.
- Может использоваться для задач классификации и оценки влияния признаков.
- Эффективен на больших выборках и может быстро обучаться на большом количестве признаков.
- Хорошо интерпретируется и может помочь понять важность каждого признака.

##### Минусы:

- Чувствительность к выбросам и шуму в данных.
- Требует выполнения предположения о линейной зависимости между признаками и целевой переменной, что может быть проблематично в некоторых случаях.
- Не учитывает взаимодействие между признаками, что может привести к недооценке важности некоторых признаков.
- Может быть склонен к переобучению, если в модели слишком много признаков или коэффициенты признаков сильно отличаются друг от друга.
- Неустойчив к мультиколлинеарности. Алгоритм логистической регрессии может быть неустойчив к мультиколлинеарности между признаками, что может привести к неопределенности в оценке весов модели.

В целом, алгоритм логистической регрессии имеет много преимуществ, таких как простота реализации, хорошая обобщающая способность и возможность вероятностной интерпретации, но также имеет ограничения, связанные с линейностью модели и чувствительностью к выбросам и мультиколлинеарности.

#### 4.2.10 Реализация алгоритма в Python

В библиотеке Scikit-learn в модуле linear\_model также реализован класс **LogisticRegression** для логистической регрессии.

Класс LogisticRegression имеет следующие параметры:

- **penalty:** строка, указывающая тип регуляризации ('l1', 'l2', 'elasticnet', None). По умолчанию равен 'l2', что означает регуляризацию L2.
- **dual:** булевый параметр, указывающий, должна ли быть решена двойственная задача оптимизации. По умолчанию равен False.
- **tol:** вещественное число, задающее критерий остановки. По умолчанию равен False.
- **C:** коэффициент регуляризации, положительное вещественное число. По умолчанию равен 1e-4.

- **fit\_intercept**: булевый параметр, указывающий, должно ли применяться смещение (intercept/bias). По умолчанию равен True.
- **intercept\_scaling**: вещественное число, указывающее, должен ли масштаб смещения быть умножен на С. По умолчанию равен 1.
- **class\_weight**: строка или словарь, указывающий, должно ли быть использовано взвешивание по классам (вес задается в зависимости от относительной частоты класса). По умолчанию равен None.
- **random\_state**: число или объект RandomState, указывающий, должен ли использоваться случайный генератор для воспроизводимости результатов работы модели. По умолчанию равен None.
- **n\_jobs**: количество параллельных задач, которые используются для расчета. По умолчанию равен None, что означает использование всех доступных процессоров.

Класс LogisticRegression имеет методы:

- **fit(X, y)** для обучения модели на данных X и y
- **predict(X)** для предсказания целевых значений для новых данных X
- **score(X, y)** для получения точности модели
- **get\_params()** для получения параметров модели соответственно
- **predict\_proba(X)**, который возвращает вероятности принадлежности классу для новых данных X.

#### 4.2.11 Вопросы для самопроверки

С помощью какой функции в логистической регрессии происходит отображение вещественных значений на отрезок (0, 1)?

1. тангенс
2. сигмоида
3. логистическая функция

Правильные ответы: 2

Какой функционал ошибки может использоваться для обучения логистической регрессии?

1. MSE
2. Логистический
3. Логарифмический

Правильные ответы: 2, 3

Выберите плюсы алгоритма логистической регрессии:

1. Эффективен на больших выборках и может быстро обучаться на большом количестве признаков.
2. Чувствительность к выбросам и шуму в данных.
3. Хорошо интерпретируется и может помочь понять важность каждого признака.
4. Требует выполнения предположения о линейной зависимости между признаками и целевой переменной, что может быть проблематично в некоторых случаях.

5. Простота реализации и понимания. Это один из самых простых алгоритмов машинного обучения.
6. Не учитывает взаимодействие между признаками, что может привести к недооценке важности некоторых признаков.
7. Может использоваться для задач классификации и оценки влияния признаков.
8. Может быть склонен к переобучению, если в модели слишком много признаков или коэффициенты признаков сильно отличаются друг от друга.

Правильные ответы: 1, 3, 5, 7

#### 4.2.12 Резюме по разделу

**Алгоритм логистической регрессии** - это метод машинного обучения, который используется для классификации объектов. Алгоритм предсказывает вероятность принадлежности объекта к определенному классу, основываясь на линейной комбинации значений признаков и коэффициентов регрессии, а также сигмоидальной функции, которая переводит вещественные значения на отрезок (0, 1).

**Коэффициенты регрессии** - это параметры модели, которые определяют веса признаков в предсказании класса объекта. Чтобы определить коэффициенты, используется функционал ошибок, называемый **логистической функцией потерь**.

**Основными преимуществами логистической регрессии являются простота и интерпретируемость модели**, а также возможность использования для классификации бинарных и многоклассовых задач. Недостатками алгоритма являются чувствительность к выбросам и нелинейным зависимостям между переменными.

## 4.3 Метод опорных векторов

**Цель занятия:** ученик может применить метод опорных векторов для решения задач классификации на подготовленных и неподготовленных данных.

**План занятия:**

- Визуальная демонстрация алгоритма
- Описание алгоритма
- Подготовка данных для алгоритма
- Процесс обучения
- Оценка качества алгоритма
- Интерпретация признаков с помощью алгоритма
- Модификации алгоритма
- Область применения алгоритма
- Плюсы и минусы алгоритма
- Реализация алгоритма в Python

### 4.3.1 Визуальная демонстрация алгоритма

Метод опорных векторов (англ. Support Vector Machine, SVM) - это алгоритм машинного обучения, использующийся для классификации. Он основывается на поиске гиперплоскости в пространстве признаков, которая наилучшим образом разделяет данные на два класса. В случае SVM мы ищем гиперплоскость, которая максимизирует зазор (отступ между двумя классами данных). Зазор (опорный вектор) в контексте SVM - это расстояние от гиперплоскости до ближайшей точки каждого класса. Визуальная демонстрация алгоритма представлена на иллюстрации 4.9.

Давайте разберемся с тем, что изображено на иллюстрации 4.9. На рисунках 4.9.1-4.9.2 отображена суть SVM - поиск гиперплоскости в пространстве признаков, которая наилучшим образом разделяет данные на два класса (оптимальный отступ). Однако такая безусловная оптимизация (Hard-margin SVM) не приспособлена для работы с объектами, которые классифицируются неправильно. Для таких случаев вводятся дополнительные переменные  $\epsilon_i$ , которые разрешают на некоторых объектах делать отступ меньше единицы и коэффициент С, которые позволяет выбирать, что важнее: сделать отступ шире либо сделать точность модели лучше (Рисунки 4.9.3)-4.9.4. На рисунках 4.9.5-4.9.6 показан так называемый Kernel trick в SVM: для линейно неразделимых выборок можно увеличить размерность данных для того, чтобы данные стали линейно разделимыми.

### 4.3.2 Описание алгоритма

Дадим формальное определение алгоритма SVM. SVM (англ. Support Vector Machine) - это алгоритм машинного обучения для задач классификации и регрессии. Он находит оптимальную гиперплоскость, которая разделяет два класса данных с наибольшим зазором (margin) между ними и наименьшим количеством ошибок при классификации.

Рассмотрим SVM для линейно разделимых выборок.

Hard-margin SVM - это метод машинного обучения, который находит линейную гиперплоскость, разделяющую два класса данных в линейно разделимом случае. Он предполагает, что существует идеальная разделяющая гиперплоскость, которая не допускает никаких ошибок классификации на обучающем наборе данных.

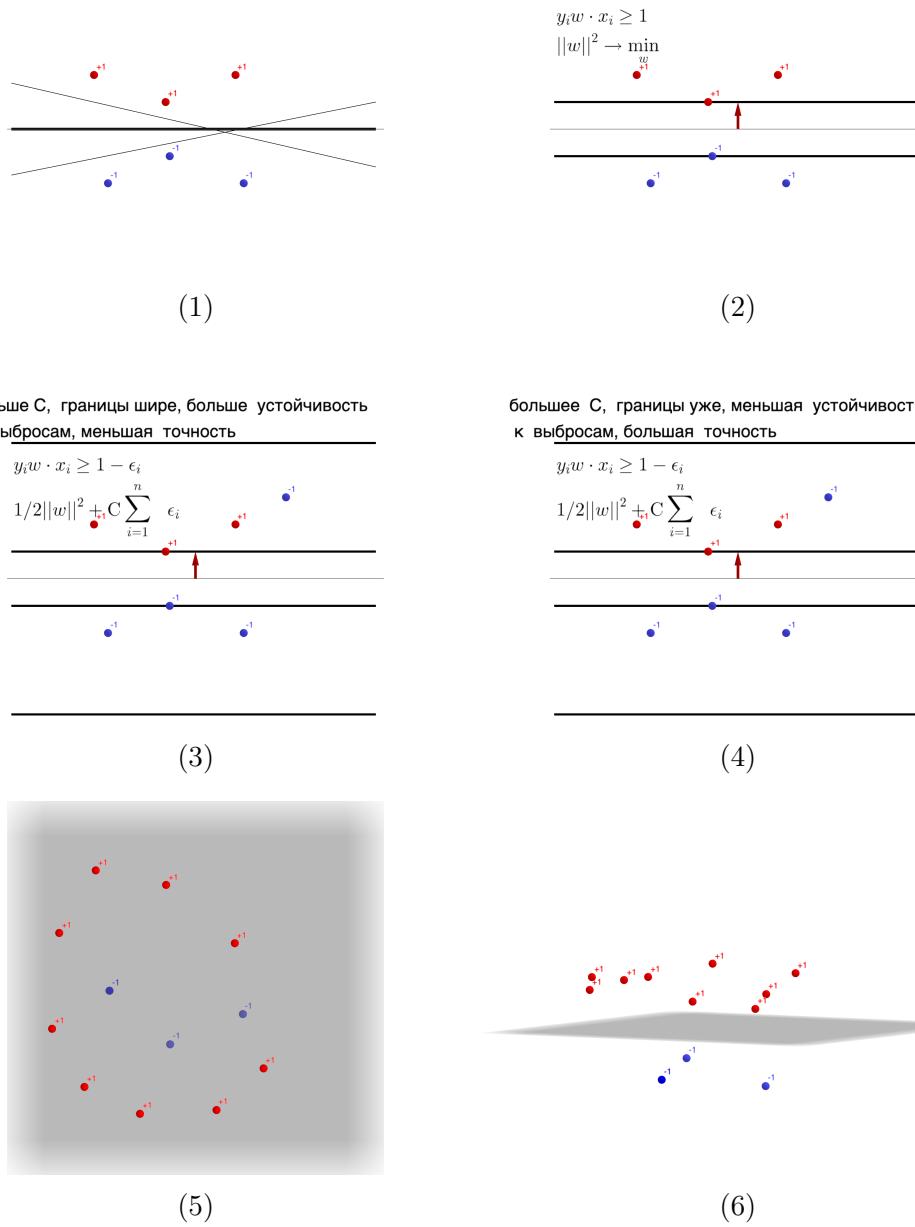


Рисунок 4.9: Визуальная демонстрация метода опорных векторов

Формально, пусть дано обучающее множество  $X$  с  $n$  элементами, где каждый элемент  $x$  представлен вектором признаков, и метки классов  $y_i$ , где  $y_i$  принимает значение 1 или -1, обозначающее класс, к которому относится  $i$ -й элемент.

Тогда, Hard-margin SVM ищет линейную гиперплоскость  $f(x) = w^T x$ , которая разделяет два класса данных с наибольшим зазором (margin) между ними и отступом:  $y_i(w \cdot x_i + w_0) > 0$ . Зазор определяется как расстояние от гиперплоскости до ближайших элементов каждого

класса. Отступ от гиперплоскости вычисляется по формуле:

$$\frac{|w \cdot x_i + w_0|}{\|w\|}$$

Соответственно, зазор вычисляется как:

$$\min_{x_i \in X} \frac{|w \cdot x_i + w_0|}{\|w\|}$$

После этого происходит нормирование весов путем деления их на минимальное значение отступа. После такой нормировки минимальное значение отступа будет равно 1:

$$\min_{x_i \in X} \frac{|w \cdot x_i + w_0|}{\|w\|} = 1$$

Можно вернуться к зазору классификатора. Так как  $\|w\|$  не зависит от  $i$ ,  $\|w\|$  выносится за знак минимума,  $\min_{x_i \in X} |w \cdot x_i + w_0|$  равен 1 после нормировки, соответственно зазор равен:

$$\min_{x_i \in X} \frac{|w \cdot x_i + w_0|}{\|w\|} = \frac{\min_{x_i \in X} |w \cdot x_i + w_0|}{\|w\|} = \frac{1}{\|w\|}$$

Соответственно, у нас появляется ряд требований к модели, на основе которых можно определить условную оптимизацию для модели:

- $y_i(w \cdot x_i + w_0) > 0$
- максимальный зазор для классификатора:  $\max_w \frac{1}{\|w\|}$  при условии, что  $|w \cdot x_i + w_0| \geq 1$

Максимизация  $\max_w \frac{1}{\|w\|}$  эквивалентна минимизации  $\min_w \|w\|$ .

В итоге получается задача условной оптимизации для SVM (Формула 4.13).

$$\begin{cases} \|w\|^2 \rightarrow \min_w (\|w\|^2 \text{ берется в квадрате для облегчения вычисления производных}) \\ |w \cdot x_i + w_0| \geq 1 \end{cases} \quad (4.13)$$

Решение этой задачи оптимизации находит вектор весов  $w$  и смещение  $b$ , которые определяют разделяющую гиперплоскость и минимизируют норму вектора весов.

Таким образом, Hard-margin SVM ищет идеальную гиперплоскость, которая разделяет данные на два класса, и не допускает никаких ошибок классификации на обучающем наборе данных. Однако, такой подход может быть чувствителен к выбросам и не работать в случае, когда данные линейно неразделимы.

В случае, если выборка линейно неразделима, условие  $|w \cdot x_i + w_0| \geq 1$  невалидно. Для этого случая используется Soft-margin SVM.

Soft-margin SVM — это модификация классического Hard-margin SVM, которая позволяет решать задачи классификации, учитывая наличие шумовых или выбросовых данных в обучающей выборке. В отличие от классического SVM, где решающая граница строго разделяет классы, в Soft-margin SVM допускаются некоторые ошибки классификации за счет допуска нарушения ограничений на расстояние от объектов до разделяющей границы (margin).

Формально, Soft-margin SVM решает задачу оптимизации, минимизируя следующий функционал (Формула 4.14):

$$\begin{cases} y_i w \cdot x_i \geq 1 - \epsilon_i \\ \frac{1}{2} \|w\|^2 + C \sum_{i=1}^n \epsilon_i \rightarrow \min_w \end{cases}$$

где  $y_i \in \{-1, 1\}$  - класс объекта,  $x_i$  - вектор его признаков,

$w$  - вектор весов,  $\epsilon_i$  - дополнительные переменные, обозначающие степень нарушения ограничений на margin для каждого объекта,

$C$  - гиперпараметр, определяющий вес ошибки классификации в оптимизационной задаче.

(4.14)

Таким образом, в Soft-margin SVM допускаются объекты, которые находятся внутри margin или даже на неправильной стороне разделяющей границы, при этом степень нарушения ограничений контролируется параметром  $C$ . Большее значение  $C$  приводит к более строгой классификации, тогда как меньшее значение  $C$  допускает большее количество ошибок классификации.

### 4.3.3 Подготовка данных для алгоритма

Перед применением алгоритма SVM необходимо выполнить следующие шаги подготовки данных:

- **Очистка данных:** удаление неполных или некорректных записей, заполнение пропущенных значений и преобразование данных в формат, подходящий для анализа.
- **Нормализация данных:** приведение данных к общему масштабу, чтобы каждый признак имел одинаковый вклад в анализ. Нормирование признаков является важным шагом в предобработке данных перед использованием SVM. Это связано с тем, что SVM оценивает веса (коэффициенты) каждого признака в модели, чтобы предсказать целевую переменную. Если масштабы разных признаков значительно отличаются друг от друга, то оценка весов может быть смешена в сторону признаков с большими значениями. Это может привести к неверным выводам и плохому качеству предсказания. Обычно используются два основных способа нормирования:
  - Стандартизация: признаки масштабируются так, чтобы их среднее значение было равно 0, а стандартное отклонение было равно 1.
  - Нормализация: признаки масштабируются так, чтобы их значения находились в диапазоне от 0 до 1.

После нормирования признаков, каждый из них будет иметь примерно одинаковый вклад в предсказание целевой переменной, что позволит SVM более точно оценивать их веса и делать более точные прогнозы.

- **Разбиение данных на обучающую и тестовую выборки:** обучающая выборка используется для обучения модели, а тестовая выборка - для оценки ее точности.

### 4.3.4 Процесс обучения

Обучение SVM с использованием градиентного спуска включает в себя следующие шаги:

1. **Формирование обучающего набора данных.** Подготавливаем обучающий набор данных, содержащий объекты, каждый из которых имеет некоторые признаки и относится к одному из двух классов.
2. **Выбор функции потерь.** В SVM обычно используют функцию потерь hinge loss, которая выражается следующим образом:  $L(y_i, \mathbf{w}) = \max(0, 1 - y_i(\mathbf{w}^T \mathbf{x}_i))$  где  $y_i$  - метка класса для объекта  $i$ ,  $\mathbf{w}$  - параметры гиперплоскости, а  $\mathbf{x}_i$  - вектор признаков объекта  $i$ .
3. **Определение оптимальных параметров.** SVM имеет гиперпараметр  $C$ , который определяет, насколько много ошибок классификации допускает SVM. В данном случае его можно интерпретировать как коэффициент регуляризации, который позволяет контролировать сложность модели. Параметр  $C$  может быть настроен с помощью кросс-валидации.
4. **Определение градиента.** В случае hinge loss градиент выражается следующим образом:  $\frac{\partial L(y_i, \mathbf{w})}{\partial \mathbf{w}} = \begin{cases} -y_i \mathbf{x}_i, & \mathbf{w}^T \mathbf{x}_i > 1 \\ 0, & \text{otherwise} \end{cases}$
5. **Обновление параметров.** Для обновления параметров  $\mathbf{w}$  и  $b$  используется формула градиентного спуска:  $\mathbf{w} \leftarrow \mathbf{w} - \alpha \frac{\partial L}{\partial \mathbf{w}}$  где  $\alpha$  - скорость обучения, которая определяет, как быстро меняются параметры.
6. Определение условия останова. Обучение останавливается, когда достигается некоторый критерий останова, например, когда достигается максимальное количество итераций или когда функция потерь перестает уменьшаться на заданном уровне точности.
7. **Повторение шагов 4-6.** Шаги 4-6 повторяются до тех пор, пока модель не достигнет заданного уровня точности или не будет выполнен критерий останова.
8. **Построение гиперплоскости.** После обучения SVM, гиперплоскость, разделяющая два класса, может быть построена из параметров  $\mathbf{w}$ .

### 4.3.5 Оценка качества алгоритма

Для оценки качества алгоритма SVM часто используют следующие метрики:

- **accuracy:**  $\frac{TP+TN}{TP+TN+FP+FN}$
- **precision:**  $\frac{TP}{TP+FP}$
- **recall:**  $\frac{TP}{TP+FN}$
- **F1:**  $2 * \frac{\text{precision}*recall}{\text{precision}+\text{recall}}$

### 4.3.6 Интерпретация признаков с помощью алгоритма

Интерпретация признаков в SVM заключается в понимании того, как каждый признак влияет на зависимую переменную в модели. Это может помочь понять, какие признаки наиболее важны для прогнозирования целевой переменной и как они связаны с ней.

Наиболее простым способом оценить влияние того или иного признака - посмотреть на весовой коэффициент при признаке. Коэффициенты признаков в SVM представляют величину и направление влияния каждого признака на зависимую переменную. Если значение признака большое, можно сказать, что его значение также большое (при условии нормализованных данных). Если коэффициент положительный, то увеличение значения признака будет приводить к увеличению значения зависимой переменной. Если коэффициент отрицательный, то

увеличение значения признака будет приводить к уменьшению значения зависимой переменной.

Важно отметить, что интерпретация признаков в SVM должна быть осуществлена с осторожностью, поскольку влияние каждого признака на принятие решения может зависеть от остальных признаков и от их взаимодействия. Кроме того, при использовании ядерных SVM, интерпретация признаков может быть затруднена, поскольку признаковое пространство может быть преобразовано в другое пространство, где трудно понять, какие именно признаки оказывают влияние на решение.

#### 4.3.7 Модификации алгоритма

Существует несколько модификаций алгоритма SVM, которые могут быть применены в зависимости от задачи и данных. Некоторые из них:

- **Soft-margin SVM:** Soft-margin SVM разрешает некоторым объектам попадать внутрь границы разделения классов (маргинала), что снижает требования к разделимости данных. Это позволяет создать более гибкую границу разделения, которая может лучше обобщать на новые данные, которые могут быть не идеально разделимыми. Вместо того, чтобы минимизировать только ошибки классификации, как в Hard-margin SVM, Soft-margin SVM минимизирует ошибки классификации и штрафует за нарушение границы маргинала.
- **SVM с ядрами:** SVM с ядрами позволяет перевести нелинейно разделимые данные в более высокую размерность пространства, где данные могут быть линейно разделимыми. Вместо того, чтобы искать гиперплоскость, разделяющую данные в исходном пространстве признаков, SVM с ядрами ищет гиперплоскость в новом пространстве, созданном путем применения функции ядра к исходным данным. Некоторые из наиболее распространенных ядер SVM включают в себя:
  - Линейное ядро: простейшее ядро, которое используется для линейной разделимости классов.
  - Полиномиальное ядро: используется для увеличения размерности пространства данных и построения нелинейных границ.
  - RBF (Radial Basis Function) ядро: самое распространенное ядро, которое создает нелинейную разделяющую гиперплоскость.
  - Sigmoid ядро: используется для задач классификации с нелинейной разделимостью.
- **Multi-class SVM:** Multi-class SVM позволяет обрабатывать данные с несколькими классами. Одним из подходов является использование метода "один против всех" в котором SVM обучается для каждого класса, чтобы отличать его от всех остальных классов.
- **Incremental SVM:** Incremental SVM позволяет обучать SVM на больших объемах данных, постепенно добавляя новые объекты и/или признаки к обучающей выборке. Это удобно, если данные поступают постепенно или если вычислительные ресурсы ограничены.
- **Online SVM:** Online SVM позволяет обучать SVM на потоке данных, который может быть бесконечным или динамическим. Это достигается путем последовательного обновления параметров SVM по мере получения новых данных.

#### 4.3.8 Область применения алгоритма

Некоторые из основных областей применения алгоритма SVM включают в себя:

- **Классификация текстов:** SVM может быть использован для классификации текстовых данных, таких как электронные письма или отзывы на продукты, на положительные и отрицательные классы.
- **Биоинформатика:** SVM может быть использован для классификации белков, распознавания генов и прогнозирования свойств биомолекул.
- **Компьютерное зрение:** SVM может быть использован для обнаружения объектов на изображении, распознавания лиц, классификации изображений и т.д.
- **Финансовый анализ:** SVM может быть использован для прогнозирования цен на акции и классификации инвестиционных продуктов.
- **Интернет-реклама:** SVM может быть использован для классификации пользователей в зависимости от их поведения в интернете и предсказания того, какие рекламные объявления будут наиболее эффективны для конкретного пользователя.
- **Анализ данных:** SVM может быть использован для кластеризации и классификации данных, таких как данные о клиентах, данные о продажах и данные о производственных процессах.
- **Медицинская диагностика:** SVM может быть использован для классификации медицинских данных, таких как изображения СТ и MRI, для определения наличия или отсутствия определенного заболевания.

В целом, алгоритм SVM является мощным инструментом машинного обучения, который может использоваться для решения различных задач классификации в различных областях.

#### 4.3.9 Плюсы и минусы алгоритма

Алгоритм SVM имеет ряд преимуществ и недостатков.

##### Плюсы:

- Эффективность: SVM хорошо справляется с выборкой большого объема и с большим количеством признаков.
- Хорошая обобщающая способность: SVM может обучаться на небольшом количестве данных, не переобучаясь.
- Гибкость: SVM может использоваться с различными типами ядер, что позволяет адаптироваться к различным задачам.
- Высокая точность: SVM может давать высокую точность при решении задач классификации.

##### Минусы:

- Чувствительность к выбросам: SVM чувствителен к выбросам в данных, что может привести к неправильным результатам.
- Не подходит для несбалансированных данных: если данные несбалансированы, то SVM может давать неправильные результаты.
- Выбор ядра: выбор подходящего ядра для конкретной задачи может быть сложной задачей.
- Не подходит для большого числа классов: SVM не всегда подходит для задач с большим числом классов.

В целом, алгоритм SVM является мощным инструментом машинного обучения, который имеет свои преимущества и недостатки. При выборе алгоритма необходимо учитывать особенности конкретной задачи и обеспечивать подходящую настройку параметров SVM.

### 4.3.10 Реализация алгоритма в Python

В библиотеке Scikit-learn также реализован класс **SVC** для SVM. Класс SVC имеет следующие параметры:

- **C**: коэффициент регуляризации. Меньшие значения C создают более широкие разделяющие гиперплоскости, позволяющие большему количеству наблюдений нарушать ограничения. По умолчанию C=1.
- **kernel**: ядро, используемое в SVM. Поддерживаются линейное ядро, радиальное ядро базисной функции (RBF), полиномиальное ядро и другие. По умолчанию используется RBF.
- **degree**: степень полиномиального ядра. Используется только если kernel='poly'. По умолчанию degree=3.
- **gamma**: коэффициент ядра RBF. Большие значения gamma приводят к более точному соответствию обучающим данным, но могут приводить к переобучению. По умолчанию gamma='scale', что означает, что оно будет равно  $1 / (\text{n\_features} * \text{X.var}())$ , где X - обучающая выборка.
- **coef0**: свободный член, используемый в ядрах полинома и сигмоиды. По умолчанию coef0=0.0.
- **shrinking**: булевый параметр, указывающий, должен ли использоваться алгоритм уменьшения границ. По умолчанию shrinking=True.
- **probability**: булевый параметр, указывающий, должны ли выдаваться оценки вероятности для классификации. По умолчанию probability=False.
- **tol**: критерий останова оптимизации. По умолчанию tol=1e-3.
- **max\_iter**: максимальное количество итераций. По умолчанию max\_iter=-1, что означает, что нет ограничения на количество итераций.

Класс SVC также имеет методы **fit(X, y)** для обучения модели на данных X и y, а также метод **predict(X)** для предсказания целевых значений для новых данных X. Кроме того, класс SVC имеет методы **score(X, y)** и **get\_params()** для получения оценки точности модели и параметров модели соответственно.

### 4.3.11 Вопросы для самопроверки

Какое главное отличие между SVM и логистической регрессией?

1. SVM является линейным классификатором, в то время как логистическая регрессия - нелинейным
2. SVM стремится найти гиперплоскость, максимально разделяющую классы, в то время как логистическая регрессия моделирует вероятности принадлежности к классам
3. SVM может работать только с двумя классами, в то время как логистическая регрессия может работать с многоклассовыми задачами
4. SVM может использовать различные ядра для нелинейного разделения классов, в то время как логистическая регрессия не имеет такой возможности

Правильные ответы: 2

Какую роль играет параметр C в SVM?

1. Он определяет ширину зазора между гиперплоскостью и ближайшими объектами разных классов.
2. Он определяет, сколько объектов может оказаться на неверной стороне гиперплоскости.
3. Он определяет, насколько сильно модель будет штрафовать за ошибки на обучающей выборке.
4. Он определяет, какую функцию потерь будет использовать модель.

Правильные ответы: 3

Выберите плюсы алгоритма SVM:

1. Хорошая масштабируемость на больших датасетах.
2. Высокая точность классификации.
3. Способность работать с данными, не являющимися линейно разделимыми с помощью ядер
4. Возможность качественно работать с большим количеством признаков.
5. Подходит для решения задач регрессии.
6. Не требует предварительной обработки данных.

Правильные ответы: 1, 2, 3, 4

#### 4.3.12 Резюме по разделу

**Алгоритм SVM (Support Vector Machine)** - это метод машинного обучения, который используется для классификации. Алгоритм строит гиперплоскость в пространстве признаков, которая разделяет объекты разных классов.

**Опорные вектора (Support Vectors)** - это объекты обучающей выборки, которые лежат ближе всего к разделяющей гиперплоскости. Опорные вектора используются для определения гиперплоскости и оптимизации функции потерь.

Основными преимуществами SVM являются эффективность в высокоразмерных пространствах, хорошая обобщающая способность и возможность работы с нелинейными данными. Недостатками алгоритма являются сложность выбора правильного ядра и параметров модели, а также чувствительность к выбросам.

Также существуют модификации алгоритма, такие как SVM с ядерными функциями, которые позволяют работать с нелинейными данными, SVM с мягким зазором (Soft-margin SVM), которая позволяет работать с неидеальными данными и SVM с несколькими классами, которые позволяют классифицировать объекты на более чем два класса.

### 4.4 Резюме по модулю

В этом модуле мы рассмотрели такие темы, как линейная классификация, логистическая регрессия, а также с метод опорных векторов. Линейная классификация отличается от задачи регрессии тем, что в ней необходимо предсказать метку класса или вероятность класса, а не вещественное число. Существует две разновидности задач классификации: бинарная классификация и многоклассовая классификация. Мы также рассмотрели функции потерь для задач классификации: логистическая функция потерь, логарифмическая функция потерь, функция потерь Хинджа.

Также были рассмотрены две модели линейной классификации: логистическая регрессия и метод опорных векторов. В логистической регрессии используется сигмоида для преобразования вещественного скалярного произведения вектора весов на вектор признаков в диапазон (0, 1). В методе опорных векторов (SVM) мы ищем гиперплоскость, которая максимизирует зазор (отступ между двумя классами данных).



# Глава 5

## Линейная классификация. Логистическая регрессия. Метод опорных векторов.

### 5.1 Деревья решений

**Цель занятия:** В этом занятии мы познакомимся с деревьями решений. После прохождения занятия ученики смогут понимать основные принципы работы моделей на основе решающих деревьев.

Ранее мы рассматривали метрические (Метод ближайших соседей) и линейные (линейная регрессия, логистическая регрессия, SVM) модели. Теперь давайте поговорим о новом классе моделей, которые хорошо справляются с нелинейными зависимостями — это решающие деревья.

Метрические и линейные модели предполагают линейную независимость признаков, что далеко не всегда хорошо. Например, в задаче предсказания цены на недвижимость увеличение расстояния от метро далеко не всегда будет означать уменьшение стоимости квадратного метра: если мы рассмотрим дополнительный признак, площадь объекта недвижимости, то для некоторых объектов увеличение расстояния до метро и увеличение площади объекта будет означать, что этот объект относится к загородной недвижимости, соответственно, цена на квадратный метр в такой недвижимости может и повыситься. В метрических и линейных моделях можно попытаться строить новые признаки на основе комбинаций двух или нескольких признаков, однако это влечет снижение интерпретируемости модели и повышение ее вычислительной сложности. Поэтому используется другой подход — алгоритм решающего дерева, который во многом похож на процедуру принятия решения человеком-экспертом. Рассмотрим этот подход подробнее.

#### 5.1.1 Описание алгоритма решающего дерева

Решающее дерево - это древовидная модель машинного обучения, которая представляет собой дерево, где каждый узел представляет тест на значение признака, а ветви представляют возможный результат этого теста. Для теста, какой путь (ребро) в дереве нужно выбрать, используются логические правила. Логические правила (также их называют предикатами или индикаторами) в решающих деревьях - это условия, которые определяют, какие действия должны быть выполнены на каждом узле дерева. Каждое правило состоит из предиката, который представляет собой условие на признаки и соответствующей операции принятия решения. Например, для классификации объектов на основе признаков, правило может быть

записано как «Если значение признака  $X_1$  меньше 5, а значение признака  $X_2$  больше 10, то идем в левую ветвь/объект принадлежит классу А, иначе идем в правую ветвь/объект принадлежит классу В». В данном случае предикатом является «значение признака  $X_1$  меньше 5, а значение признака  $X_2$  больше 10», а операция принятия решения - «объект принадлежит классу А». Логические правила в решающих деревьях могут с одним условием или содержать несколько условий. Они помогают определить, какой путь в дереве следует выбрать для получения правильного результата. Условие на предикат можно записать так:

$$[x_i \leq t],$$

где  $t$  - порог по определенному признаку.

В целом, ничего не мешает использовать в предикатах сложные модели и условия, но в этом обычно нет необходимости, и в качестве предиката берут сравнение значения по одному определенному признаку с порогом.

Давайте опишем устройство решающего дерева на примере наглядного набора данных в виде двух подков (Рисунок 5.1). Самая верхняя вершина называется корневой. Другие вер-

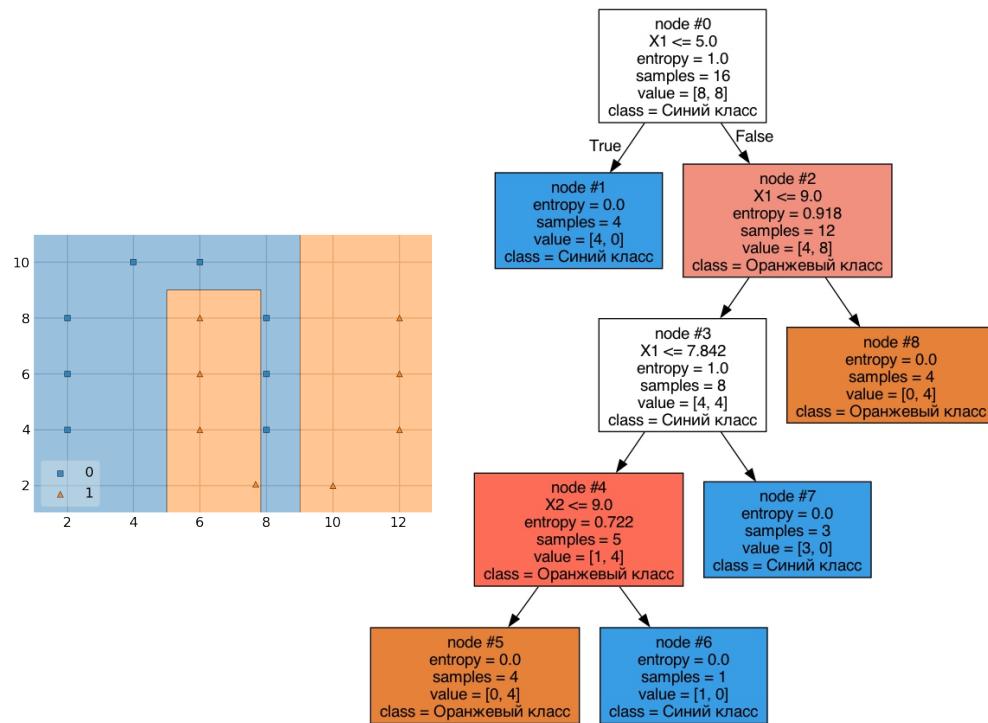


Рисунок 5.1: Построение решающего дерева на примере набора данных в виде двух подков

шины с предикатами называются внутренними. Логическое правило для корневой вершины формулируется так: « $X_1 \leq 5.0$ ». Всего в этой вершине рассматривается 16 объектов ( $\text{samples} = 16$ ), поделенных на 2 класса ([8, 8]). Если предиктор « $X_1 \leq 5.0$ » для объекта равен True, то объект отправляется в левую вершину, иначе — в правую. После обработки объектов в левой вершине станет 4 объекта класса «Синий класс». Левая вершина называется листовой (для неё уже нет предикторов, только итоговый прогноз для части выборки). В дереве решений это будет тупиковая, конечная ветка, которая не ведёт к новым условиям — она выглядит как лист на дереве и поэтому зовётся листовой. В листовой вершине присваивается метка самого популярного класса в случае классификации или вероятность класса (равная доле

объектов класса в вершине), либо выдается константный прогноз (усредненный по объектам из обучающей выборки, попавшим в данную вершину) в случае регрессии.

Для правого поддерева процесс деления выборки с помощью предикатов продолжится, пока все вершины не станут листовыми. Построение разделяющей поверхности для данного примера показано на рисунке 5.2. По сути, решающее дерево разбило все пространство признаков на прямоугольные области, в каждой определенной области объекту присваивается определенный класс.

### 5.1.2 Критерии информативности

Очень важным является то, как выбираются предикаты. Для этого используются критерии информативности. Критерии информативности — это метрики, которые используются при построении моделей машинного обучения для оценки качества разделения данных на разные классы или значения (в случае регрессии). В контексте деревьев решений, критерии информативности используются для выбора наилучшего разбиения признаков на узлах дерева.

Основными критериями информативности являются:

- для классификации — энтропия и критерий Джини
- для регрессии — дисперсия

**Энтропия** является мерой неопределенности в системе и может быть выражена следующим образом (Формула 5.1):

$$H(X) = - \sum_{i=1}^C p_i \log_2(p_i), \quad (5.1)$$

где  $X$  — это признак,  $C$  — множество классов,

$p_i$  — вероятность появления класса  $i$ .

Чем больше энтропия, тем больше неопределенности в системе. Если все значения переменной  $X$  равновероятны, то энтропия будет максимальной, а если все значения имеют одну и ту же величину, то энтропия будет минимальной.

**Критерий Джини (Gini impurity)** — это еще один критерий информативности, используемый для построения деревьев решений. Он также измеряет неопределенность выборки и может использоваться вместо энтропийного критерия.

Формула для расчета критерия Джини выглядит следующим образом (Формула 5.2):

$$Gini(X) = 1 - \sum_{i=1}^C p_i^2, \quad (5.2)$$

где  $X$  — это признак,  $C$  — множество классов,

$p_i$  — вероятность появления класса  $i$ .

Чем меньше значение критерия Джини, тем более однородными являются объекты в выборке. И наоборот, чем больше значение критерия Джини, тем менее однородными являются объекты в выборке. При использовании критерия Джини в деревьях решений выбирается тот признак, который позволяет наиболее эффективно уменьшить значение критерия Джини при разбиении выборки на подгруппы.

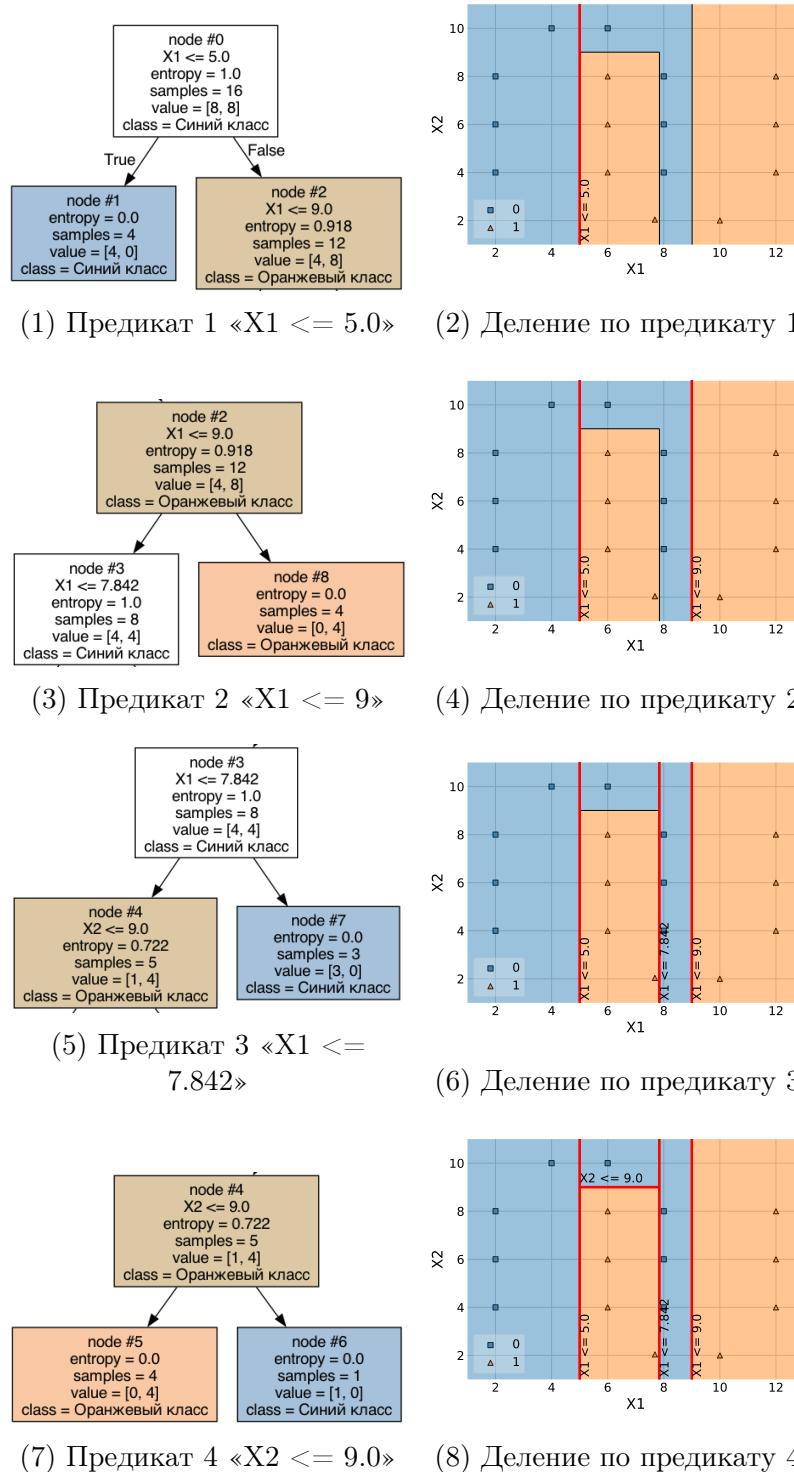


Рисунок 5.2: Построение разделяющей поверхности для набора данных в виде двух подков

При построении дерева решений мы пытаемся разбить данные на подгруппы, где каждая группа будет иметь меньшую энтропию или меру Джини, чем исходная группа. Таким образом, критерий информативности в деревьях может быть определен как разность между

энтропией/критерием Джини исходного узла и суммой энтропий/критериев Джини полученных поддеревьев.

Можно продемонстрировать смысл критериев информативности диаграммой (Рисунок 5.3).

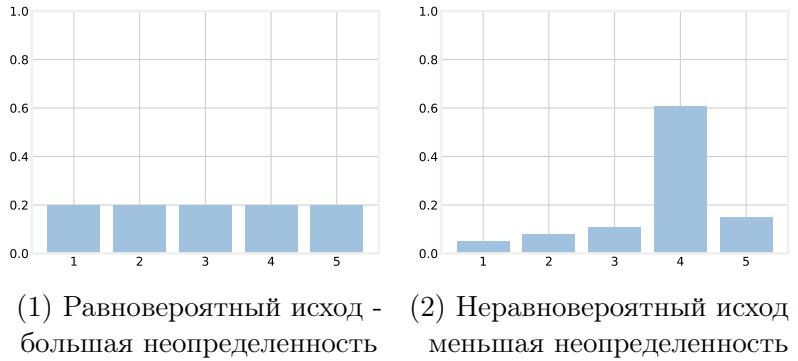


Рисунок 5.3: Смысл энтропии и критерия Джини

**Дисперсия** является критерием информативности для задач регрессии в деревьях решений. Критерий дисперсии позволяет оценить насколько хорошо определенный признак разделяет целевую переменную.

Для расчета критерия дисперсии используется следующая формула (Формула 5.3):

$$Var(X) = \frac{1}{n} \sum_{i=1}^n (x_i - \bar{x})^2,$$

где  $n$  - количество объектов в выборке, (5.3)

$x_i$  - значение целевой переменной для  $i$ -го объекта,

$\bar{x}$  - среднее значение целевой переменной в выборке.

При разбиении выборки на подгруппы по значению определенного признака, происходит уменьшение значения дисперсии в каждой из подгрупп. Информативность признака оценивается как разница между начальным значением дисперсии и суммой дисперсий в подгруппах, взвешенной на количество объектов в каждой подгруппе.

Выбирается тот признак, который позволяет максимально уменьшить значение дисперсии при разбиении выборки на подгруппы. Этот признак становится корневым узлом дерева, а процесс разбиения продолжается для каждой подгруппы до тех пор, пока не будет достигнут критерий остановки.

### 5.1.3 Процесс построения дерева для классификации и регрессии

Давайте посмотрим, как происходит процесс построения дерева на примере энтропии. Дерево обычно строится с использованием жадного алгоритма (выбираются оптимальные для уровня признак и порог на этом признаке только на этом узле, не пытаясь просчитать несколько шагов вперед). Процесс построения дерева происходит с помощью прироста информации. **Прирост информации (Information Gain)** в деревьях решений является мерой, которая используется для определения того, какой признак лучше всего разделяет данные на различные классы или категории.

В деревьях решений признаки используются для разделения данных на более чистые группы, каждая из которых относится к определенному классу или категории. Цель состоит в том, чтобы создать дерево решений, которое максимизирует число правильно классифицированных данных.

Information Gain измеряет изменение энтропии (меры неопределенности) или критерия Джини в данных, которое происходит при разделении данных на основе конкретного признака. Таким образом, прирост информации является мерой, которая показывает, какой признак наиболее полезен для разделения данных на более чистые группы, которые максимально отличаются друг от друга по классам или категориям. Чем выше прирост информации, тем более полезен признак для разделения данных на более чистые группы.

Соответственно, процесс построения дерева включает в себя следующие шаги:

1. Начните с корневого узла, содержащего все обучающие данные.
2. Для каждого признака, рассчитайте энтропию (количество неопределенности) набора данных, используя формулу энтропии (Формула 5.1).
3. Для каждого признака, рассчитайте прирост информации (Information Gain), используя формулу 5.4.

$$IG(S, A) = H(S) - \sum \frac{|S_v|}{|S|} * H(S_v),$$

где  $IG(S, A)$  - прирост информации для признака A,

$|S_v|$  - количество элементов в поднаборе данных, где значение признака A равно v,

$|S|$  - общее количество элементов в наборе данных,

$H(S_v)$  - энтропия поднабора данных, где значение признака A равно v.

(5.4)

4. Выберите признак с наибольшим приростом информации или минимальной энтропией. Разделите данные на основе выбранного признака, создавая два дочерних узла.
5. Рекурсивно повторяйте шаги 2-4 для каждого дочернего узла до выполнения некоторого условия остановки, например, максимальной глубины дерева или минимального числа объектов в узле.

Процесс построения дерева с помощью прироста информации для регрессии аналогичен процессу для классификации, но использует другую меру неопределенности. Вместо энтропии, используется дисперсия.

Вот общий процесс построения дерева с помощью прироста информации для регрессии:

1. Начните с корневого узла, содержащего все обучающие данные.
2. Для каждого признака вычислите прирост информации или изменение дисперсии, которое будет получено при разбиении данных на основе этого признака, используя формулу 5.5.

$$IG(S, A) = Var(S) - \sum \frac{|S_v|}{|S|} * Var(S_v),$$

где  $IG(S, A)$  - прирост информации для признака A,

$|S_v|$  - количество элементов в поднаборе данных, где значение признака A равно v,

$|S|$  - общее количество элементов в наборе данных,

$Var(S_v)$  - дисперсия поднабора данных, где значение признака A равно v.

(5.5)

3. Выберите признак с наибольшим приростом информации или минимальной дисперсией. Разделите данные на основе выбранного признака, создавая два дочерних узла.
4. Рекурсивно повторяйте шаги 2-4 для каждого дочернего узла до выполнения некоторого условия остановки, например, максимальной глубины дерева или минимального числа объектов в узле.

Решающее дерево — хороший инструмент для прогнозирования на данных, но этот инструмент обладает одним ярко выраженным недостатком — нестабильностью. Процесс построения дерева с помощью прироста информации может приводить к появлению неоптимальных разбиений, особенно в случае, когда имеется большое количество признаков с высокой корреляцией.

При применении модели мы обычно ожидаем, что на приблизительно одинаковых данных мы получим схожую модель. С решающими деревьями это не так. Даже при небольшом изменении выборки решающее дерево сильно изменяет разделяющую поверхность (Рисунок 5.4), что, по сути, является признаком переобучения.

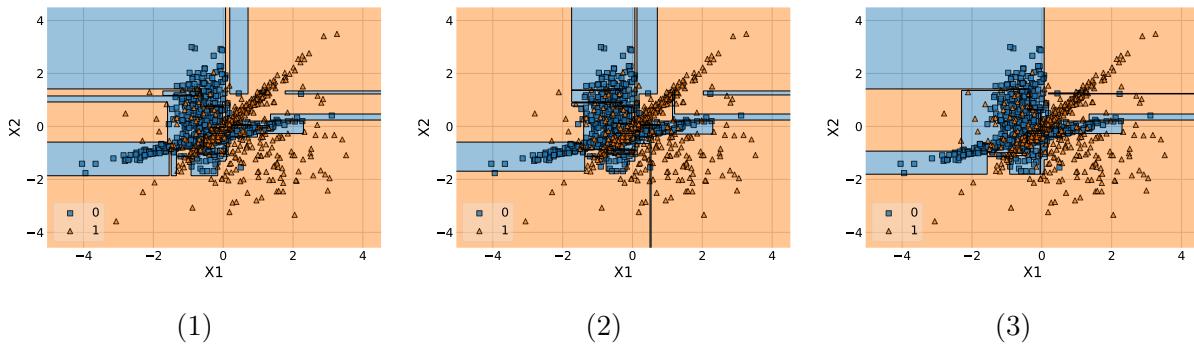


Рисунок 5.4: Изменение разделяющей поверхности при изменении 10% выборки

На рисунке 5.4 видно, что одиночное решающее дерево достаточно сильно изменяется при небольшом изменении данных.

Для уменьшения риска переобучения, можно использовать регуляризацию деревьев и композиции деревьев, такие как случайный лес (Random Forest) или градиентный бустинг деревьев (Gradient Boosted Trees).

#### 5.1.4 Регуляризация деревьев решений

Регуляризация деревьев решений - это процесс добавления дополнительных ограничений на дерево решений, чтобы уменьшить переобучение модели. Деревья решений, как правило, склонны к переобучению, если они имеют слишком много листьев и слишком глубоко ветвятся на тренировочных данных, что может привести к неспособности модели обобщать на новые данные.

Существуют несколько способов регуляризации деревьев решений:

- **Ограничение глубины дерева (max\_depth)** - это ограничение на максимальную глубину дерева. Если дерево достигает максимальной глубины, оно перестает ветвиться и превращается в лист, что предотвращает переобучение.

- **Минимальное количество элементов в листе (min\_samples\_leaf)** - это ограничение на минимальное количество обучающих элементов, которые должны быть в каждом листе дерева. Если количество элементов в листе меньше заданного значения, то он не будет разветвляться, что уменьшает переобучение.
- **Максимальное количество листьев (max\_leaf\_nodes)** - это ограничение на максимальное количество листьев в дереве. Если дерево достигает максимального количества листьев, оно не будет больше разветвляться, что предотвращает переобучение.
- **Регуляризация посредством снижения веса (weight regularization)** - в этом методе добавляется штраф к функции потерь модели за большие веса, что приводит к более простым моделям и уменьшает переобучение.
- **Подрезка деревьев (англ. Pruning)** - это процесс уменьшения размера дерева решений путем удаления частей дерева, которые не приносят значимого улучшения качества предсказаний (Рисунок 5.5). Основная идея прунинга заключается в том, что

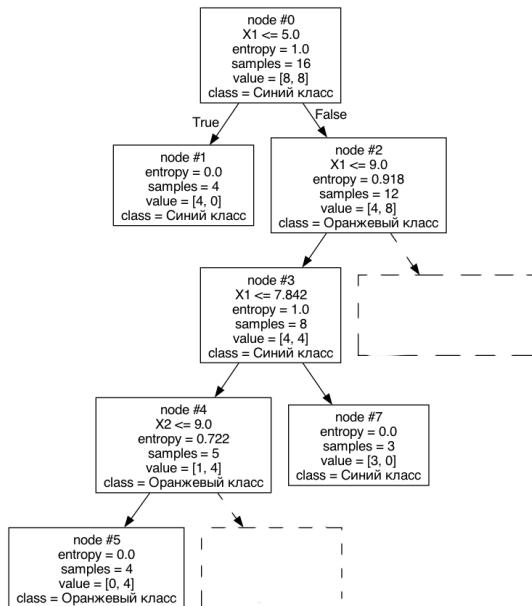


Рисунок 5.5: Подрезка деревьев

большие деревья решений могут слишком сильно подстроиться под обучающие данные и, следовательно, переобучаться. Удаление части дерева может привести к уменьшению переобучения и улучшению обобщающей способности модели.

Существуют различные методы прунинга деревьев решений, включая pre-pruning, post-pruning и reduced error pruning. Pre-pruning означает остановку роста дерева решений до достижения определенной глубины или количества листьев. Post-pruning, с другой стороны, является процессом подрезки дерева после того, как оно было полностью построено. Reduced error pruning - это метод, который удаляет узлы из дерева решений, если это приводит к уменьшению ошибок на отложенной выборке.

Применение прунинга деревьев решений позволяет улучшить обобщающую способность модели, снизить вероятность переобучения и уменьшить сложность модели. Однако важно найти правильный баланс между уменьшением размера дерева и сохранением информации, которую дерево содержит.

Бороться с переобучением также помогают композиции деревьев, они рассмотрены в следующем разделе.

### 5.1.5 Вопросы для самопроверки

В каких частях деревьев аккумулируются итоговые прогнозы модели решающего дерева?

1. Корень
2. Ребра
3. Внутренние вершины
4. Листья

Правильные ответы: 4

Какой метод используется для вычисления значения регрессии в листе у решающего дерева для нескольких объектов?

1. Среднее значение целевой переменной у всех объектов в листе
2. Медианное значение целевой переменной у всех объектов в листе
3. Среднее квадратичное отклонение целевой переменной у всех объектов в листе
4. Наиболее часто встречающееся значение целевой переменной у всех объектов в листе

Правильные ответы: 1

Какие методы используется для выбора оптимального разделения в решающем дереве в задачах классификации?

1. Дисперсия
2. Критерий Джини
3. Энтропия
4. Метод опорных векторов

Правильные ответы: 2, 3

### 5.1.6 Резюме по разделу

В этом занятии мы познакомились с деревьями решений, узнали, что такое критерии информативности, и то, как строится дерево решения для задач классификации и регрессии.

## 5.2 Композиции деревьев

**Цель занятия:** ученик может применять смещение и разброс в моделях машинного обучения, композиции деревьев для повышения устойчивости моделей на основе деревьев решений.

### 5.2.1 Подходы к построению композиций

В машинном обучении агрегирование нескольких моделей для получения более точного и устойчивого прогноза носит название «композиция моделей» или «ансамблирование». В целом, композиция моделей на основе деревьев решений может значительно повысить качество предсказаний в задачах классификации и регрессии (Рисунок 5.6).

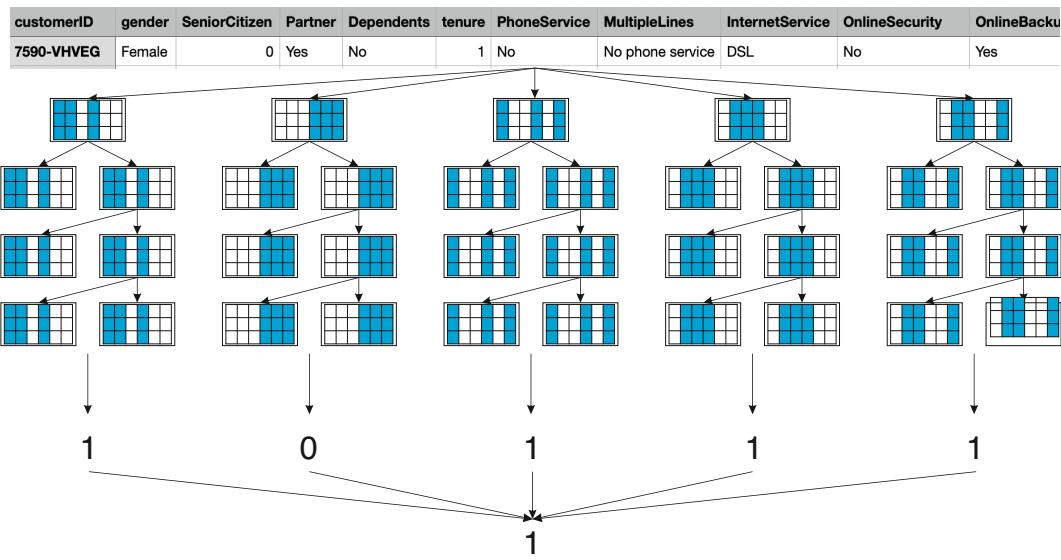


Рисунок 5.6: Композиция деревьев

Наиболее простым способом определения класса для композиции деревьев является голосование по большинству (англ. Majority vote), когда метка класса присваивается по относительному большинству отдельных моделей, предсказавших её. При одинаковом количестве голосов существуют разные стратегии: случайный выбор, приоритет определенных классов и пр. Для регрессии наиболее очевидный способ агрегирования — это усреднение предсказаний моделей из композиции.

На рисунке 5.7 можно увидеть, что применение композиции деревьев повышает устойчивость модели.

**Бэггинг (Bootstrap Aggregating) и бустинг (Boosting)** - это два популярных подхода к построению композиций моделей машинного обучения.

**Бэггинг** - это метод построения композиции моделей, в котором **обучается несколько независимых моделей** на случайных подмножествах обучающих данных с повторениями, а затем усредняются их предсказания для получения окончательного предсказания. Бэггинг позволяет снизить дисперсию модели, т.к. усреднение предсказаний моделей позволяет уменьшить эффект переобучения. Примерами алгоритмов бэггинга являются случайный лес и бэггинг деревьев решений.

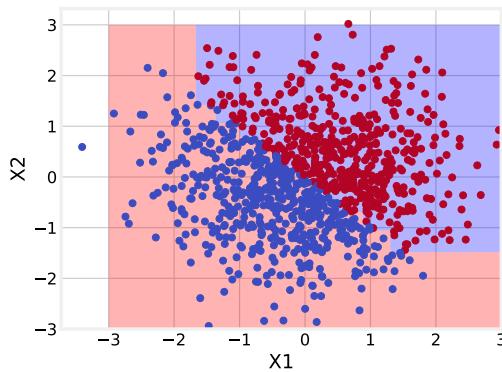


Рисунок 5.7: Повышение устойчивости предсказаний при использовании композиции деревьев

**Бустинг** - это метод построения композиции моделей, в котором **модели обучаются последовательно**, каждая из моделей корректирует ошибки предыдущей на обучающих данных. Поэтому каждая следующая модель фокусируется на тех объектах, на которых предыдущие модели ошибались. Бустинг позволяет снизить смещение модели, т.к. последующие модели корректируют ошибки предыдущих моделей. Примерами алгоритмов бустинга являются градиентный бустинг и AdaBoost.

Общий подход при использовании бэггинга и бустинга заключается в создании ансамбля моделей, который является более точным, чем каждая из отдельных моделей. Однако бэггинг и бустинг имеют различные преимущества и недостатки, и выбор подхода зависит от специфики данных и задачи.

В этом занятии мы обсудим такой подход, как бэггинг.

### 5.2.2 Бэггинг

В бэггинге (Bootstrap Aggregating) выбор случайных подмножеств объектов и признаков происходит с целью уменьшения корреляции между отдельными моделями в композиции. Это позволяет снизить дисперсию композиции и сделать ее более устойчивой к шуму (Рисунок 5.8).

Выбор случайного подмножества объектов происходит путем генерации новых выборок из исходного набора данных методом бутстрэпа (с повторениями). Каждый раз случайно выбирается подмножество объектов, которые используются для обучения отдельной модели. Количество объектов в каждом подмножестве также может быть настроено как гиперпараметр.

Выбор случайных подмножеств признаков (факторов) также может применяться в бэггинге для уменьшения корреляции между отдельными моделями. Однако, в отличие от выбора случайных подмножеств объектов, выбор случайных подмножеств признаков не производится на каждом шаге генерации новых выборок, а применяется на уровне отдельной модели. Выбор случайных подмножеств признаков может осуществляться различными способами, например, случайным выбором признаков на каждой вершине дерева при построении случайного леса или случайным выбором признаков при обучении каждой модели в композиции. Однако стоит помнить, что выбор подмножества признаков может негативно отразиться на качестве моделей, так как важность признаков неравнозначна, и следует с осторожностью применять такое

age	sex	bmi	bp	s1	s2	s3	s4	s5	s6	target
0.038076	0.050680	0.061696	0.021872	-0.04223	-0.04823	0.043400	-0.002952	0.019907	-0.017646	151.0
-0.001882	-0.044642	-0.051474	-0.026328	-0.008449	-0.019163	0.074412	-0.039493	0.068332	-0.092204	75.0
0.085299	0.050680	0.044442	-0.005670	-0.045959	-0.034194	0.072358	-0.002952	0.002861	-0.029390	141.0
-0.089063	-0.044642	-0.051195	-0.036656	0.012194	0.024991	0.036038	0.033098	0.022688	0.009362	206.0
0.005383	-0.044642	-0.036385	0.021872	0.003935	0.015598	0.008142	-0.002952	-0.01988	-0.046441	135.0
-0.092695	-0.044642	-0.040696	-0.019442	-0.068991	-0.042127	0.067394	-0.041176	0.062917	-0.096346	97.0
-0.045472	0.050680	-0.047163	-0.015999	-0.040096	-0.024809	0.009779	-0.034989	-0.062917	-0.038357	138.0
0.063304	0.050680	-0.001095	0.066629	0.009620	0.108914	0.022869	0.017703	0.035816	0.032064	63.0
0.041708	0.050680	0.061696	0.040099	-0.013973	0.006202	0.023679	0.002952	0.014990	0.011149	110.0
0.070980	-0.044642	0.039682	-0.033213	-0.01257	0.014580	0.023991	0.002952	0.067737	-0.011584	310.0
-0.096328	-0.044642	-0.083508	0.080101	-0.03389	-0.090561	0.013948	0.067659	0.062917	-0.034215	101.0
0.027178	0.050680	0.075986	-0.032213	-0.07073	0.045972	0.065491	0.071210	0.096435	-0.059067	69.0
0.016281	-0.044642	-0.028840	-0.009113	-0.040321	-0.009769	0.044958	0.039493	0.030748	-0.042499	179.0
0.005383	0.050680	-0.001095	0.080101	-0.040321	-0.015719	0.002903	0.002952	0.03394	-0.013504	185.0
0.045341	-0.044642	-0.025607	-0.012556	0.017694	-0.000603	0.081775	-0.034989	0.01988	-0.075363	118.0
0.052738	0.050680	-0.018062	0.080401	0.089244	-0.107662	0.039719	0.108111	0.036060	-0.042499	171.0
0.005315	-0.044642	0.047296	0.080415	0.074574	-0.073861	0.074412	0.038480	0.052777	-0.075017	166.0
0.070769	0.050680	0.021217	0.056201	0.034206	0.049416	0.029718	0.034309	0.027364	0.001078	144.0
0.038207	-0.044642	-0.010517	-0.03656	-0.037344	-0.019476	0.023674	0.002952	-0.018114	-0.017646	97.0
0.027310	-0.044642	-0.018062	-0.040099	-0.02945	-0.011355	0.037598	0.03849	0.009443	-0.054925	168.0
0.049105	-0.044642	-0.058683	-0.043542	-0.045959	-0.043276	0.009779	0.03849	0.011897	0.013491	88.0
0.085430	0.050680	-0.022373	0.061216	-0.037344	0.026366	0.015005	0.035983	0.072113	-0.017646	49.0
0.085430	-0.044642	-0.004050	-0.009113	-0.029495	0.007767	0.022869	0.035983	0.061176	-0.015004	68.0
0.045341	0.050680	0.06608	0.01106	0.023702	0.047327	0.054446	0.072110	0.133597	0.135812	245.0
0.063353	-0.044642	0.038529	0.022885	-0.030664	-0.018859	0.006858	0.002952	0.02993	-0.049282	184.0
-0.067268	0.050680	-0.012673	-0.040099	-0.015328	0.004636	0.03812	0.043098	0.019196	-0.034215	202.0
-0.107226	-0.044642	-0.077342	-0.026228	-0.089630	-0.096109	0.026550	-0.07639	0.042571	-0.005220	137.0
-0.023677	-0.044642	0.059541	-0.040099	0.042848	-0.043389	0.011824	0.034983	0.015999	0.040435	85.0
0.052606	-0.044642	-0.021295	-0.074527	-0.040096	-0.017639	0.006858	0.035983	0.006612	-0.054925	131.0

Рисунок 5.8: Выбор случайных подмножеств объектов и признаков в бэггинге

В целом, выбор случайных подмножеств объектов или признаков является важной составляющей бэггинга и позволяет увеличить устойчивость композиции к шуму и повысить ее точность. В целом, в машинном обучении устойчивость и качество работы моделей формализуются через понятия смещения и разброса. Давайте рассмотрим их.

### 5.2.3 Смещение и разброс

Смещение и разброс - это два ключевых понятия в машинном обучении, связанные с проблемой переобучения (overfitting) и недообучения (underfitting). Во многом, это обобщение оценки недообучения и переобучения, в том числе для композиций моделей.

**Смещение (bias)** - это ошибка модели, связанная с её недостаточной способностью улавливать реальную зависимость между признаками и целевой переменной. Модель с большим смещением может быть недообучена, т.е. её способность к обобщению данных будет низкой.

**Разброс (variance)** - это ошибка модели, связанная с её чрезмерной чувствительностью к случайным шумам в данных. Модель с большим разбросом может быть переобучена, т.е. она может слишком точно подогнаться под тренировочные данные и показывать плохие результаты на новых данных.

Ошибка модели может быть разложена на составляющие (Формула 5.6).

$$\text{Ошибка} = \text{смещение}^2 + \text{разброс} + \text{шум},$$

где смещение<sup>2</sup> обозначает квадрат ошибки модели, связанной с её

недостаточной способностью улавливать реальную зависимость между признаками и целевой переменной, разброс - ошибку модели, связанную

с её чрезмерной чувствительностью к случайным шумам в данных, шум - случайную ошибку, не поддающуюся объяснению моделью.

Смысл смещения и разброса модели показан на рисунке (Рисунок 5.9). Trade-off между смещением и разбросом - это баланс между уменьшением ошибки модели, связанной с смещением, и ошибки модели, связанной с разбросом. Цель заключается в том, чтобы достичь оптимального уровня ошибки модели, который обеспечивает наилучшую способность

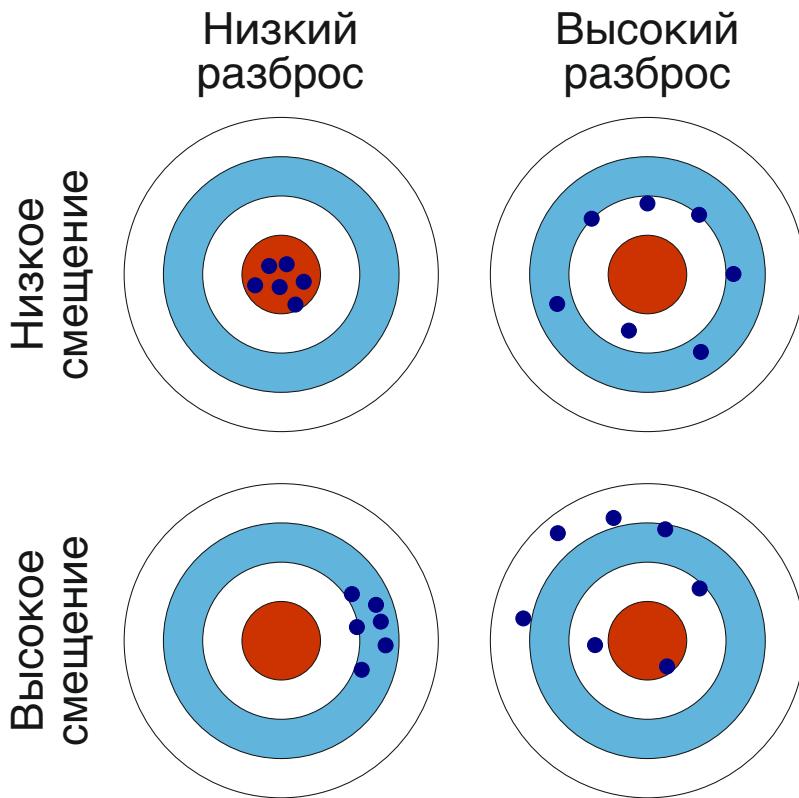


Рисунок 5.9: Отображение смысла смещения и разброса

к обобщению данных. Увеличение сложности модели может уменьшить ошибку смещения, но увеличить ошибку разброса, что может привести к переобучению. Уменьшение сложности модели может уменьшить ошибку разброса, но увеличить ошибку смещения, что может привести к недообучению (Рисунок 5.10).

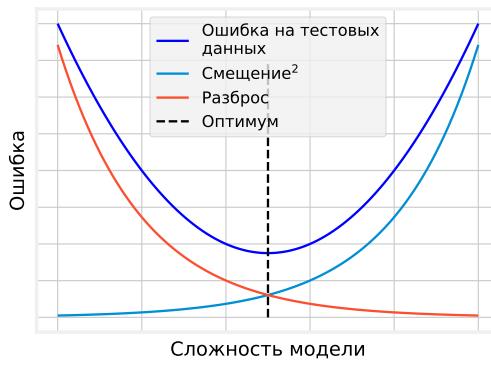


Рисунок 5.10: Баланс смещения и разброса

Бэггинг помогает уменьшить разброс, за счет использования нескольких случайных подмножеств обучающих данных и обучения независимых моделей на каждом из этих подмножеств. Каждая модель будет иметь свои сильные и слабые стороны, что позволит уменьшить разброс в итоговой модели.

С другой стороны, бэггинг не влияет на смещение, так как он все еще использует ту же функциональную форму модели на каждом подмножестве данных. Однако, в случае использования решающих деревьев в качестве базовых моделей для бэггинга, можно ожидать, что смещение также уменьшится. Это связано с тем, что решающие деревья могут описывать сложные зависимости в данных, а их комбинация позволяет учесть большее количество зависимостей.

Подбор оптимальных параметров модели (например, глубины дерева) позволяет найти баланс между смещением и разбросом, когда модель не переобучена и способна обобщать данные (Рисунок 5.11).

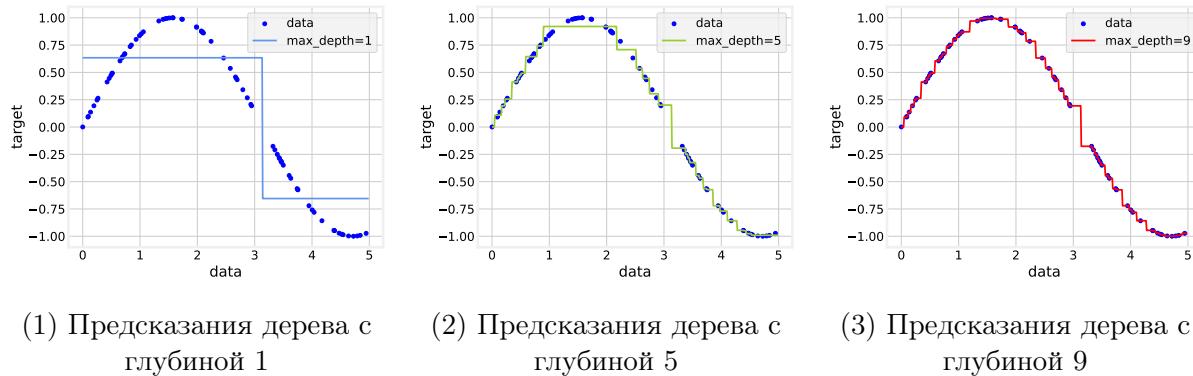


Рисунок 5.11: Подбор оптимальных параметров дерева для нахождения баланса bias-variance

При этом важно помнить, что если модели в композиции взаимосвязаны и ошибки одной модели могут быть связаны с ошибками другой модели, то использование бэггинга для уменьшения разброса может не дать значительного эффекта. Чем более связаны и похожи модели, тем меньший эффект оказывает бэггинг, и тем больше разброса остается в композиции. Следовательно, для построения композиции моделей необходимо, чтобы базовые модели были максимально независимыми и различными друг от друга. Кроме того, базовые модели также должны быть сложными, чтобы обеспечить достаточную гибкость для моделирования сложных зависимостей в данных. Эти требования к бэггингу реализованы в алгоритме случайного леса, о нем мы поговорим далее.

#### 5.2.4 Вопросы для самопроверки

Какова разница между бэггингом (Bagging) и бустингом (Boosting)?

1. Бэггинг и бустинг - это одно и то же, просто разные названия одного метода обучения.
2. Бэггинг и бустинг используют одинаковые алгоритмы, но разные функции потерь.
3. Бэггинг строит ансамбль из нескольких независимых моделей, бустинг строит ансамбль из нескольких последовательных моделей.
4. Бэггинг и бустинг - это разные методы, которые используют один и тот же алгоритм обучения, но разные наборы данных.

Правильные ответы: 3

Какой из нижеперечисленных пунктов является главной характеристикой бэггинга для решающих деревьев (один ответ)?

1. Использует множество разнородных моделей для получения одной композиции
2. Стремится уменьшить смещение модели
3. Уменьшает разброс модели
4. Использует градиентный спуск для настройки моделей

Правильные ответы: 3

Каково определение смещения (bias) в контексте концепции «bias-variance trade-off»?

1. Мера различий между предсказаниями модели и реальными значениями в тестовом наборе данных.
2. Степень изменчивости предсказаний модели при разных значениях входных данных.
3. Мера ошибки модели, вызванная недостаточной ее сложностью.
4. Сумма квадратов отклонений предсказаний модели от среднего значения на обучающем наборе данных.

Правильные ответы: 3

### 5.2.5 Резюме по разделу

Одиночное решающее дерево имеет склонность к нестабильности, когда даже при небольшом изменении выборки предсказания модели значительно меняются. Для решения этой проблемы применяют композиции моделей, такие как бэггинг и бустинг.

Бэггинг на деревьях (Bagging of Trees) - это метод ансамблирования, в котором используется множество независимых решающих деревьев для получения одной композиционной модели. Каждое дерево строится на основе выборки из обучающего набора данных, которая формируется случайным образом с повторениями. При формировании композиции каждое дерево получает равный вес в соответствии с принципом голосования. Бэггинг на деревьях помогает уменьшить разброс (variance) модели и повысить ее устойчивость к выбросам.

Смещение и разброс - это два ключевых понятия в статистике и машинном обучении, которые связаны с ошибками предсказаний модели.

Смещение (bias) - это мера того, насколько среднее значение предсказаний модели отличается от правильных значений целевой переменной. Разброс (variance) - это мера того, насколько различаются предсказания модели для разных наблюдений из обучающей выборки. Идеальной моделью является та, у которой низкое смещение и низкий разброс. Однако, уменьшение смещения может привести к увеличению разброса, а уменьшение разброса может привести к увеличению смещения. Поэтому важно найти баланс между смещением и разбросом при разработке модели.

## 5.3 Случайный лес

**Цель занятия:** ученик может применить алгоритм случайного леса для решения задач классификации на подготовленных и неподготовленных данных.

**План занятия:**

- Визуальная демонстрация алгоритма
- Подготовка данных для алгоритма
- Процесс обучения
- Оценка качества алгоритма
- Интерпретация признаков с помощью алгоритма
- Процесс применения
- Область применения алгоритма
- Плюсы и минусы алгоритма
- Модификации алгоритма
- Реализация алгоритма в Python

### 5.3.1 Визуальная демонстрация алгоритма

Случайный лес (Random Forest) - это алгоритм машинного обучения, который используется для задач классификации и регрессии. Он является композицией решающих деревьев, где каждое дерево обучается независимо друг от друга на разных случайных подвыборках данных и с разными случайными подмножествами признаков в каждой вершине.

В процессе построения случайного леса сначала случайным образом выбирается подмножество обучающих данных (bootstrap-выборка) и подмножество признаков в каждой вершине. Затем на этом подмножестве данных строится решающее дерево, используя выбранные признаки. Этот процесс повторяется множество раз, что приводит к созданию ансамбля решающих деревьев. При предсказании каждое дерево выдает свой ответ, а итоговый ответ случайного леса определяется путем голосования или усреднения ответов всех деревьев. Визуальная демонстрация алгоритма представлена на рисунке 5.12.

Давайте разберемся с тем, что изображено на иллюстрации 5.12. Мы строим композицию из 5 деревьев. Далее **для каждого нового узла с предикатами выбирается свое подмножество признаков**. Как мы упоминали, проблемой классического бэггинга может быть то, что подмножество признаков выбирается для каждого дерева — соответственно, в это подмножество могут не попасть какие-нибудь важные признаки. Поэтому в random forest подмножества выбираются для каждой вершины с предикатами. Для нового объекта строится предсказание по каждому дереву, затем происходит голосование по большинству для классификации или усреднение для регрессии.

### 5.3.2 Подготовка данных для алгоритма

Подготовка данных является важным шагом для построения эффективной модели случайного леса. Подготовка данных для алгоритма может включать в себя следующие шаги:

- Работа с выбросами. Случайный лес не чувствителен к выбросам, но они могут негативно повлиять на качество модели, особенно если они присутствуют в большом количестве. Соответственно, можно применить различные методы обработки выбросов.

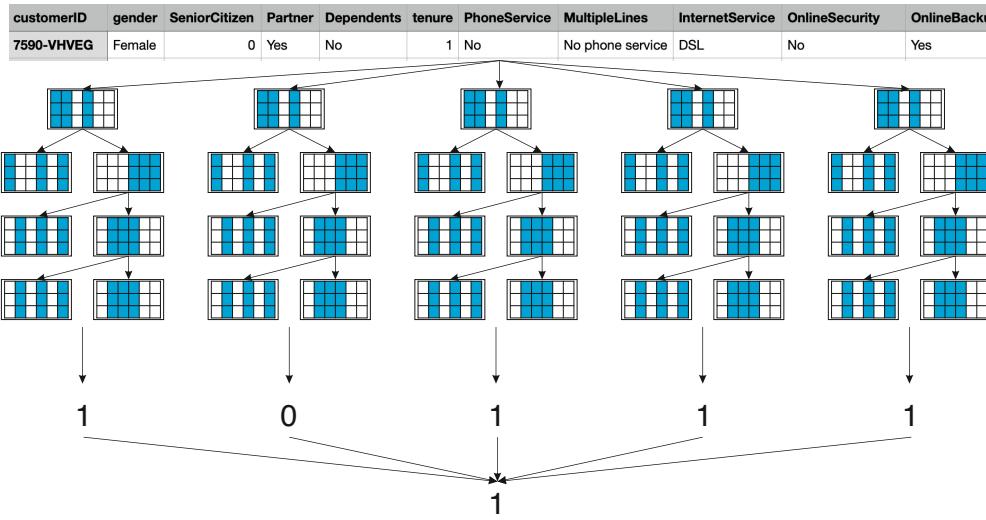


Рисунок 5.12: Визуальная демонстрация алгоритма случайный лес

- Преобразование категориальных переменных в числовые. Случайный лес не может работать с категориальными переменными напрямую, поэтому их необходимо преобразовать в числовые значения.
- Отбор признаков. Если в наборе данных есть большое количество признаков, можно использовать методы отбора признаков, такие как метод главных компонент или методы выбора признаков на основе значимости, чтобы уменьшить количество признаков и улучшить производительность модели.
- Разделение данных на обучающую и тестовую выборки.

### 5.3.3 Процесс обучения

Процесс обучения случайного леса включает в себя следующие шаги:

1. Для каждого из деревьев в ансамбле производится случайный выбор подмножества данных из обучающей выборки. Этот шаг называется подвыборкой с заменой (bootstrapping).
2. Случайный выбор подмножества признаков для каждого нелистового узла в дереве. Обычно для каждого узла выбирается  $\sqrt{d}$  признаков, где  $d$  - общее количество признаков.
3. Построение деревьев. Для каждой выборки данных и подмножества признаков строится дерево решений. Деревья строятся до определенной глубины или до тех пор, пока каждый лист не содержит минимальное количество примеров.

4. Прогнозирование. Для каждого нового примера производится прогноз с помощью каждого дерева. Класс, выбранный большинством деревьев, становится итоговым прогнозом.
5. Тюнинг гиперпараметров. Для улучшения качества модели можно провести тюнинг гиперпараметров, таких как количество деревьев, глубина деревьев, размер подвыборки и т.д.
6. Повторение. Шаги 1-5 повторяются несколько раз для получения стабильного прогноза.

В результате обучения случайного леса получается ансамбль деревьев, каждое из которых принимает решение на основе случайной подвыборки данных и признаков. Это помогает уменьшить переобучение и увеличить стабильность модели. Кроме того, случайный лес может обрабатывать данные с большим количеством признаков и работать с несбалансированными данными.

### 5.3.4 Оценка качества алгоритма

Для оценки качества алгоритма random forest часто используют следующие метрики:  
Классификация:

- **accuracy:**  $\frac{TP+TN}{TP+TN+FP+FN}$
- **precision:**  $\frac{TP}{TP+FP}$
- **recall:**  $\frac{TP}{TP+FN}$
- **F1:**  $2 * \frac{precision*recall}{precision+recall}$

Регрессия:

- **MSE**
- **MAE**

### 5.3.5 Интерпретация признаков с помощью алгоритма

Случайный лес может рассчитывать важность признаков на основе их вклада в уменьшение критерия ошибки, например, величиной критерия информативности Gini или приростом информации (Information Gain). Важность признака может быть использована для оценки вклада признака в предсказание целевой переменной. Признаки с более высокой важностью считаются более значимыми для модели.

### 5.3.6 Процесс применения

Random Forest создает ансамбль решающих деревьев, который демонстрирует лучшую точность, чем каждое отдельное дерево. Он также способен автоматически обрабатывать пропущенные значения, шум и выбросы в данных. Этот алгоритм легко распараллеливается, что делает его эффективным в использовании на больших датасетах.

Предсказание для нового объекта делается на основе агрегирования предсказаний всех деревьев в случайном лесу (например, путем голосования).

### 5.3.7 Область применения алгоритма

Некоторые из основных областей применения алгоритма случайного леса включают в себя:

- **Финансы:** анализ кредитных рисков, определение мошенничества, прогнозирование финансовых показателей.
- **Медицина:** диагностика заболеваний, прогнозирование риска заболеваний, анализ медицинских данных.
- **Реклама:** прогнозирование эффективности рекламы, анализ поведения потребителей.
- **Интернет-магазины:** рекомендательные системы, прогнозирование продаж, сегментация аудитории.
- **Обработка естественного языка:** классификация текстов, анализ тональности.
- **Изображения и видео:** распознавание объектов, классификация изображений, анализ видео.
- **Промышленность:** мониторинг и управление качеством продукции, прогнозирование отказов оборудования.
- **Транспорт и логистика:** прогнозирование спроса на перевозки, оптимизация маршрутов, анализ логистических данных.

В целом, алгоритм случайного леса может быть полезным в любой области, где требуется классификация или регрессия на основе большого количества признаков и где есть достаточно данных для обучения модели.

### 5.3.8 Плюсы и минусы алгоритма

Алгоритм случайного леса имеет ряд преимуществ и недостатков.

#### Плюсы:

- Хорошая точность предсказаний: случайный лес обладает хорошей точностью при классификации и регрессии, особенно при использовании большого количества деревьев в лесу.
- Устойчивость к переобучению: случайный лес имеет способность устранять переобучение благодаря случайности выборки и выбора признаков, что делает его устойчивым к выбросам и шуму в данных.
- Высокая скорость обучения: обучение случайного леса можно распараллелить, что позволяет быстро обрабатывать большие объемы данных.
- Возможность оценки важности признаков: случайный лес может определить наиболее важные признаки для классификации или регрессии.
- Универсальность: алгоритм случайного леса применим для решения широкого круга задач машинного обучения.

#### Минусы:

- Неинтерпретируемость: каждое дерево в случайном лесу можно интерпретировать, но в целом, сам алгоритм не обеспечивает понимания, как именно происходит принятие решения.
- Зависимость от выбора гиперпараметров: необходимо подбирать гиперпараметры, такие как количество деревьев и максимальная глубина дерева, для достижения наилучшей производительности.

- Неэффективность для работы с большим количеством категориальных признаков: алгоритм случайного леса не всегда хорошо работает с большим количеством категориальных признаков, поскольку в таком случае деревья могут становиться слишком глубокими.

В целом, алгоритм случайного леса является мощным инструментом машинного обучения с высокой точностью предсказаний и способностью устранять переобучение, но требует тщательной настройки гиперпараметров и может быть неэффективным при работе с большим количеством категориальных признаков.

### 5.3.9 Модификации алгоритма

Существует несколько модификаций алгоритма случайного леса, которые могут быть применены в зависимости от задачи и данных. Некоторые из них:

- **Random Decision Forest (RDF):** RDF - это расширение стандартного случайного леса, которое использует случайное количество деревьев для каждого класса. Это может улучшить предсказания в ситуациях, когда классы имеют различные размеры.
- **Balanced Random Forest (BRF):** BRF - это модификация случайного леса, в которой используется взвешенное голосование деревьев, чтобы уравновесить несбалансированные данные. Каждое дерево в BRF обучается на случайном подмножестве данных, сбалансированном по классам.

### 5.3.10 Реализация алгоритма в Python

В библиотеке Scikit-learn реализован класс **RandomForestClassifier** для алгоритма случайного леса. Основные параметры класса RandomForestClassifier:

- **n\_estimators:** количество деревьев в лесу. По умолчанию n\_estimators=100.
- **criterion:** функция для измерения качества разделения. Поддерживаются два критерия: «gini» для критерия Джини и «entropy» для энтропийного критерия. По умолчанию criterion=«gini».
- **max\_depth:** максимальная глубина каждого дерева в лесу. По умолчанию max\_depth=None, что означает, что деревья разрастаются до тех пор, пока все листья не будут чистыми (т.е. содержат только элементы одного класса), или пока не будет достигнуто минимальное количество элементов для разделения.
- **min\_samples\_split:** минимальное количество элементов, необходимое для того, чтобы узел мог быть разделен на два подузла. По умолчанию min\_samples\_split=2.
- **min\_samples\_leaf:** минимальное количество элементов, которые должны быть в листьях дерева. По умолчанию min\_samples\_leaf=1.
- **min\_weight\_fraction\_leaf:** минимальная доля суммы весов (всех элементов), которая должна быть в листьях дерева. По умолчанию min\_weight\_fraction\_leaf=0.0.
- **max\_features:** количество признаков, которые должны быть рассмотрены при каждом разделении. Поддерживаются следующие значения: «auto» (выбор  $\sqrt{n\_features}$  признаков), «sqrt» (то же, что «auto»), «log2» (выбор  $\log_2(n\_features)$  признаков), None (выбор всех признаков), целое число (выбор конкретного количества признаков) или доля (выбор доли от общего количества признаков). По умолчанию max\_features=«auto».

- **max\_leaf\_nodes:** максимальное количество листьев в дереве. По умолчанию max\_leaf\_nodes=None, что означает, что нет ограничения на количество листьев.

Класс **RandomForestRegressor** для решения задач регрессии имеет практически те же параметры, за исключением:

- **criterion:** функция для измерения качества разделения. Поддерживаются критерии: «squared\_error», «absolute\_error», «friedman\_mse», «poisson». По умолчанию criterion=«squared\_error».

Класс **RandomForestClassifier/RandomForestRegressor** имеют методы **fit(X, y)** для обучения модели на данных X и y, а также метод **predict(X)** для предсказания целевых значений для новых данных X. Кроме того, классы **RandomForestClassifier/RandomForestRegressor** имеют методы **score(X, y)** и **get\_params()** для получения оценки точности модели и параметров модели соответственно.

### 5.3.11 Вопросы для самопроверки

Как происходит выбор подмножества признаков в случайном лесе?

1. Подмножество признаков выбирается для всей композиции деревьев
2. Подмножество признаков выбирается для каждого дерева
3. Подмножество признаков выбирается для каждой нелистовой вершины в дереве
4. Используются все признаки из набора данных

Правильные ответы: 3

Как можно использовать алгоритм случайного леса для задач регрессии?

1. Использовать критерий Джини для разбиения по предикатам и голосование по большинству для прогнозирования
2. Использовать энтропию для разбиения по предикатам и голосование по большинству для прогнозирования
3. Использовать MSE для разбиения по предикатам и усреднение для прогнозирования

Правильные ответы: 3

Выберите плюсы алгоритма случайного леса:

1. Возможность оценки важности признаков: случайный лес может определить наиболее важные признаки для классификации или регрессии.
2. Высокая вычислительная сложность: поскольку в случайном лесе много деревьев, обработка данных может занять длительное время.
3. Высокая скорость обучения: обучение случайного леса можно распараллелить, что позволяет быстро обрабатывать большие объемы данных.
4. Зависимость от выбора гиперпараметров: необходимо подбирать гиперпараметры, такие как количество деревьев и максимальная глубина дерева, для достижения наилучшей производительности.
5. Неинтерпретируемость: каждое дерево в случайном лесу можно интерпретировать, но в целом, сам алгоритм не обеспечивает понимания, как именно происходит принятие решения.

6. Неэффективность для работы с большим количеством категориальных признаков: алгоритм случайного леса не всегда хорошо работает с большим количеством категориальных признаков, поскольку в таком случае деревья могут становиться слишком глубокими.
7. Универсальность: алгоритм случайного леса применим для решения широкого круга задач машинного обучения.
8. Устойчивость к переобучению: случайный лес имеет способность устранять переобучение благодаря случайности выборки и выбора признаков, что делает его устойчивым к выбросам и шуму в данных.
9. Хорошая точность предсказаний: случайный лес обладает хорошей точностью при классификации и регрессии, особенно при использовании большого количества деревьев в лесу.

Правильные ответы: 1, 3, 7, 8, 9

### 5.3.12 Резюме по разделу

**Алгоритм случайного леса** основан на идее комбинирования нескольких деревьев решений, обученных на разных подмножествах признаков в каждой нелистовой вершине и подмножествах объектов. В результате получается ансамбль деревьев, который позволяет уменьшить влияние переобучения и повысить точность классификации или регрессии.

Одним из ключевых преимуществ случайного леса является его способность работать с большим количеством признаков и наблюдений, что делает его подходящим для обработки сложных и многоуровневых данных. Кроме того, случайный лес может быть использован для интерпретации важности признаков, что может быть полезно для понимания вклада каждого признака в процесс принятия решения.

## 5.4 Резюме по модулю

В этом модуле мы рассмотрели такие темы, как деревья решений, композиции деревьев и случайный лес.

Дерево решений - это алгоритм машинного обучения, который используется для решения задач классификации и регрессии. Он представляет собой древовидную структуру, в которой каждый узел представляет тест на определенном признаке, а каждая ветвь - возможный результат этого теста.

Деревья решений могут быть очень эффективными в решении задач, так как они просты в интерпретации и могут использоваться как для категориальных, так и для количественных данных. Кроме того, они могут быть применены для любого количества классов или значений целевой переменной.

Однако деревья решений могут также страдать от переобучения, особенно если они слишком глубокие или содержат много признаков. Этот недостаток может быть устранен с помощью различных методов, включая подрезку дерева, ограничение глубины и использование ансамблевых методов, таких как случайный лес.

# Глава 6

## Бустинг. Adaboost. Градиентный бустинг. XGBoost. LightGBM. Catboost.

### 6.1 Бустинг

Мы уже рассмотрели бэггинг как способ построения композиции моделей на деревьях.

Бэггинг (bootstrap aggregating) представляет собой метод ансамблирования, при котором строится несколько независимых моделей на разных подвыборках обучающего набора данных, а затем их результаты усредняются. Бэггинг помогает бороться с разбросом (variance), то есть с тем, что результаты модели могут сильно варьироваться в зависимости от выбора обучающей выборки.

Бустинг (boosting) также представляет собой метод ансамблирования, но он построен на итеративном улучшении одной базовой модели путем добавления новых моделей, каждая из которых исправляет ошибки предыдущих (Рисунок 6.1). Бустинг помогает бороться со смеще-

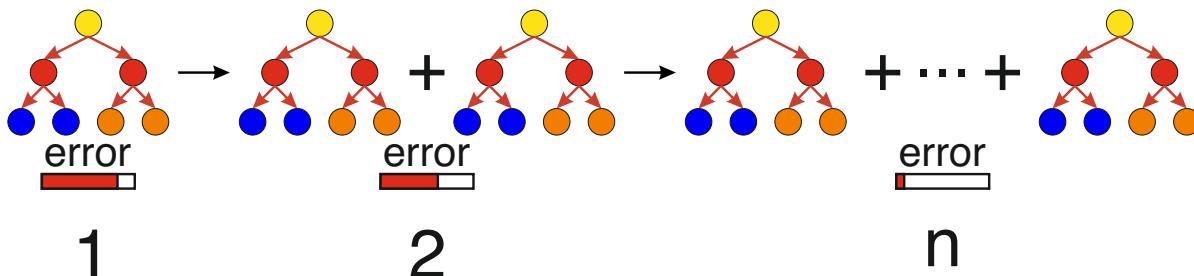


Рисунок 6.1: Визуализация принципа бустинга

нием (bias), то есть с тем, что модель может недообучаться и давать неточные предсказания на новых данных.

Таким образом, бэггинг и бустинг являются двумя различными подходами к решению проблемы ансамблирования моделей, и каждый из них борется с определенным типом ошибки модели.

Основные отличия между бэггингом и бустингом:

- Бэггинг использует ансамбль независимых моделей, каждая из которых обучается на случайной подвыборке данных. Бустинг использует последовательный ансамбль моделей, каждая из которых учитывает ошибки предыдущей модели.

- Бэггинг может использоваться с любым алгоритмом обучения, тогда как бустинг обычно используется с деревьями решений.
- Бэггинг может использоваться для уменьшения дисперсии модели, тогда как бустинг может использоваться для уменьшения как дисперсии, так и смещения модели.

Можно отметить следующие преимущества бустинга:

- Бустинг использует последовательное обучение, что позволяет каждой новой модели учитывать ошибки предыдущих моделей, что может привести к более точным прогнозам.
- Бустинг может обнаруживать сложные зависимости в данных, тогда как бэггинг склонен к созданию более простых моделей.
- Бустинг может использоваться для решения различных задач, включая классификацию, регрессию и ранжирование, тогда как бэггинг чаще используется для решения задач классификации.

В целом, бустинг может работать лучше, когда требуется более точная модель, способная обнаруживать сложные зависимости в данных, и когда задача может быть решена последовательно. Однако, бэггинг также может быть эффективным, особенно если требуется уменьшить дисперсию модели и улучшить ее устойчивость к шуму в данных.

Две наиболее популярные разновидности бустинга — это AdaBoost и градиентный бустинг (Gradient Boosting).

Основными различиями между AdaBoost и градиентным бустингом являются следующие:

- Обучение: AdaBoost обучает модель путем последовательного добавления новых моделей, которые перевзвешивают объекты, на которых допущены ошибки на предыдущих итерациях, в то время как градиентный бустинг обучает модель, используя градиентный спуск, минимизируя функцию потерь.
- Сложность моделей: AdaBoost использует простые модели, такие как деревья решений с одним разделением (decision stumps), в то время как градиентный бустинг может использовать более сложные модели, такие как деревья решений с несколькими уровнями.
- Регуляризация: градиентный бустинг поддерживает регуляризацию для уменьшения переобучения, в то время как AdaBoost не поддерживает регуляризацию.
- Шум: AdaBoost чувствителен к шуму в данных, потому что объекты с ошибками классификации получают более высокие веса, что может привести к переобучению, в то время как градиентный бустинг более устойчив к шуму.

В целом, градиентный бустинг обычно дает более высокое качество модели, чем AdaBoost, за счет использования более сложных моделей и регуляризации. Однако AdaBoost может быть быстрее и более простым в использовании, особенно в задачах с небольшим объемом данных.

Давайте разберем эти модели более подробно.

### 6.1.1 Adaboost

AdaBoost (англ. Adaptive Boosting) — это алгоритм машинного обучения, используемый для задач классификации и регрессии. Он был предложен Йоавом Фрейдом и Робертом Шапиро в 1996 году.

AdaBoost работает путем последовательного добавления «слабых» классификаторов в композицию. Каждый классификатор обучается на выборке, на которой предыдущие классификаторы допустили ошибки. При этом объекты, которые были неправильно классифицированы, получают более высокие веса, чтобы следующий классификатор мог сосредоточиться на них и исправить ошибки.

Каждый слабый классификатор представляет собой простую модель, такую как дерево решений с одним разделением (decision stump), которая предсказывает значение целевой переменной на основе одного признака. Каждый классификатор получает вес, который зависит от его точности, при этом более точные классификаторы получают больший вес. Соответственно, AdaBoost обучает набор слабых моделей на последовательно изменяющихся весах данных и комбинирует их в сильную модель.

Когда все слабые классификаторы обучены, они объединяются в одну композицию с помощью взвешенного голосования, где вес каждого классификатора зависит от его точности. Эта композиция является итоговой моделью.

Основным преимуществом AdaBoost является его способность уменьшать ошибку на обучающей выборке с каждой новой итерацией, что приводит к более высокой точности на тестовых данных. Однако он может быть чувствителен к шуму в данных и может приводить к переобучению, если слабые классификаторы слишком сложны.

Давайте рассмотрим пример с двумя классами: синими и оранжевыми метками. Для обучения AdaBoost мы будем использовать базовый алгоритм классификации - решающие деревья глубины 1, называемые пнями решений. Иллюстрация демонстрирует, как каждый новый классификатор (пень решений) настраивает веса объектов в обучающей выборке (Рисунок 6.2).

Первоначально все объекты имеют одинаковый вес. Затем мы обучаем первый классификатор и вычисляем его ошибку. Ошибка равна количеству объектов, которые были неправильно классифицированы, поделенному на общее количество объектов. Как только мы вычислили ошибку, мы можем рассчитать вес этого классификатора. Чем меньше ошибка, тем выше вес.

Затем мы пересчитываем веса объектов в обучающей выборке. Если объект был неправильно классифицирован, то его вес увеличивается. Если объект был правильно классифицирован, то его вес уменьшается. Затем мы повторяем процедуру с новым классификатором, вычисляя ошибку, вес и пересчитывая веса объектов. На каждой итерации мы добавляем новый классификатор к существующим и пересчитываем веса объектов.

После нескольких итераций AdaBoost начинает фокусироваться на объектах, которые ему трудно классифицировать. Это означает, что такие объекты получают больший вес. В конечном итоге мы получаем ансамбль классификаторов, каждый из которых специализируется на определенной области пространства признаков.

## Визуализация работы AdaBoost

Ниже приведена визуализация работы AdaBoost на наборе с двумя признаками (Рисунок 6.2).

## Формальное определение AdaBoost

Формальное определение AdaBoost выглядит так:

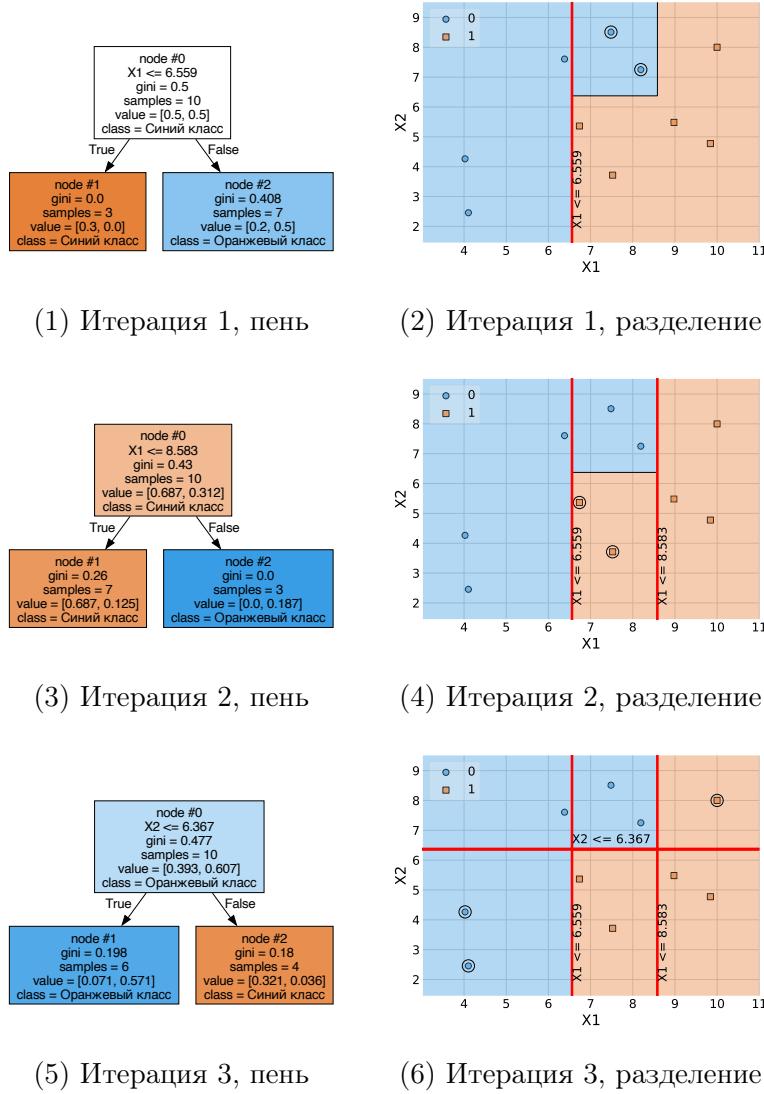


Рисунок 6.2: Визуализация работы AdaBoost

Исходно имеется обучающая выборка  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$ , где  $\mathbf{x}_i$  - вектор признаков  $i$ -го объекта, а  $y_i$  - его метка класса.

На каждой итерации  $m$  AdaBoost строит классификатор  $h_m(\mathbf{x})$  с помощью базового алгоритма обучения.

На первой итерации все объекты имеют равные веса:  $w_{1,i} = 1/n$ , где  $i = 1, \dots, n$ .

Далее, на каждой итерации  $m = 1, \dots, M$ , AdaBoost делает следующее:

1. Обучает классификатор  $h_m(\mathbf{x})$  на выборке  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_n, y_n)$  с весами  $w_{m,i}$ .
2. Вычисляет ошибку классификатора на обучающей выборке:

$$err_m = \frac{\sum_{i=1}^n w_{m,i} I(y_i \neq h_m(\mathbf{x}_i))}{\sum_{i=1}^n w_{m,i}},$$

где  $I(\cdot)$  - индикаторная функция (функция, которая принимает значение 1 при выполнении определенного условия и значение 0 в противном случае).

3. Вычисляет вес классификатора:

$$\alpha_m = \log \frac{1 - err_m}{err_m}.$$

4. Обновляет веса объектов:

$$w_{m+1,i} = w_{m,i} \exp(\alpha_m I(y_i \neq h_m(\mathbf{x}_i))),$$

где  $i = 1, \dots, n$ .

После  $M$  итераций AdaBoost объединяет классификаторы  $h_1(\mathbf{x}), \dots, h_M(\mathbf{x})$  с помощью взвешенного голосования:

$$H(\mathbf{x}) = \text{sign} \left( \sum_{m=1}^M \alpha_m h_m(\mathbf{x}) \right).$$

Здесь функция  $\text{sign}(\cdot)$  возвращает знак своего аргумента (т.е. +1 или -1), что соответствует бинарной классификации.

AdaBoost был одним из популярных алгоритмов бустинга, однако на сегодняшний момент большей популярностью пользуется градиентный бустинг.

### 6.1.2 Градиентный бустинг

Градиентный бустинг - это алгоритм машинного обучения, использующий последовательное обучение слабых моделей (например, деревьев решений), каждая из которых нацелена на исправление ошибок предыдущей модели. Ошибки модели измеряются при помощи градиента функции потерь, и каждая последующая модель обучается на остатках (разности между предсказаниями текущей модели и правильными ответами) предыдущих моделей. После обучения всех моделей, их предсказания комбинируются в итоговый ансамбль с помощью взвешенного голосования или суммирования. Градиентный бустинг является одним из наиболее мощных алгоритмов машинного обучения и широко используется в различных областях, таких как рекомендательные системы, естественный язык и другие.

Рассмотрим формальное определение задачи для градиентного бустинга. Пусть у нас есть обучающая выборка  $(x_i, y_i)$ , где  $x_i$  — входные данные, а  $y_i$  — их соответствующие метки класса или значения целевой переменной. Для решения задачи регрессии будем считать, что  $y_i$  — действительные числа, а для задачи классификации — целые числа, каждое из которых соответствует классу.

В градиентном бустинге используется модель ансамбля деревьев решений, которая представляет собой сумму  $T$  деревьев с параметрами  $\theta_j$ :

$$F(x) = \sum_{j=1}^T f_j(x; \theta_j)$$

Каждое дерево  $f_j(x; \theta_j)$  является функцией, которая принимает входные данные  $x$  и возвращает соответствующий вклад в ответ ансамбля. Цель градиентного бустинга — найти параметры  $\theta_j$  каждого дерева таким образом, чтобы ансамбль  $F(x)$  минимизировал функционал ошибки<sup>1</sup>  $L(y, F(x))$  на обучающей выборке. Для этого на каждом шаге добавляется новое дерево, которое минимизирует остаточную ошибку, вычисленную как отрицательный градиент функционала ошибки по отношению к предсказаниям текущей модели:

---

<sup>1</sup>усредненное значение функции потерь для набора данных или выборки

$$r_{ij} = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$$

$$f_j = \operatorname{argmin}_f \sum_{i=1}^n (f(x_i; \theta_j) - r_{ij})^2$$

Здесь  $r_{ij}$  — это остаточное значение для  $i$ -го объекта на  $j$ -м шаге, а  $f_j$  — это новое дерево решений, которое добавляется к ансамблю на  $j$ -м шаге.

Алгоритм продолжает добавлять новые деревья, пока не будет достигнуто заданное количество итераций или пока ошибка на валидационной выборке не перестанет уменьшаться.

Градиентный бустинг получил такое название от того, что он использует градиент функционала ошибки по предсказаниям модели для построения последовательности деревьев, которые пытаются уменьшить эту ошибку.

Когда мы строим модель, мы обычно оптимизируем функционал ошибки, который зависит от параметров модели. Для нахождения оптимальных параметров мы должны вычислить градиент этого функционала ошибки по параметрам и выполнить шаг градиентного спуска, чтобы переместиться в направлении уменьшения ошибки.

В градиентном бустинге мы также хотим уменьшить функционал ошибки, но мы делаем это не напрямую, а путем построения последовательности деревьев, каждое из которых старается уменьшить остаточную ошибку на обучающей выборке. В результате, градиентный бустинг может быть рассмотрен как метод градиентного спуска в пространстве ответов на каждой итерации.

## Градиентный бустинг для регрессии

Для задачи регрессии градиентный бустинг можно формально определить следующим образом:

Дано множество объектов  $X$  и соответствующие им значения целевой переменной  $y$ , где  $y_i \in \mathbb{R}$  для каждого  $i = 1, \dots, n$ . Требуется построить модель регрессии в виде суммы функций  $F(x) = \sum_{j=1}^M f_j(x)$ , где каждое слагаемое  $f_j(x)$  является деревом решений небольшой глубины.

Для построения модели градиентного бустинга используется функция потерь  $L(y, F)$ , которая зависит от истинных значений целевой переменной  $y$  и предсказанных значений  $F(x)$  на каждом объекте  $x$ . Обычно используется квадратичная функция потерь:

$$L(y, F) = \frac{1}{2}(y - F(x))^2$$

Градиент функции потерь на каждом объекте  $x_i$  определяется как производная функции потерь по предсказанию на этом объекте:

$$\nabla L(y_i, F) = -(y_i - F(x_i))$$

На каждой итерации  $m = 1, \dots, M$  мы добавляем новое дерево  $f_m$  в модель, который приближает градиент функции потерь  $\nabla L(y_i, F)$  на каждом объекте. Таким образом, новая модель выглядит как:

$$F_m(x) = F_{m-1}(x) + \gamma_m f_m(x)$$

где  $\gamma_m$  - шаг градиентного спуска на  $m$ -ой итерации, который вычисляется как:

$$\gamma_m = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, F_{m-1}(x_i) + \gamma f_m(x_i))$$

То есть, мы ищем значение шага  $\gamma_m$ , которое минимизирует функционал ошибки на текущей итерации. На каждой итерации мы строим дерево  $f_m$  с помощью градиентного спуска по квадратичной функции ошибки:

$$f_m = \arg \min_f \sum_{i=1}^n (r_i - f(x_i))^2$$

где  $r_i = -\nabla L(y_i, F)$  - остатки модели на  $i$ -ом объекте на текущей итерации.

Таким образом, на каждой итерации мы добавляем новое дерево, которое приближает остатки модели на текущей итерации и уменьшает функционал ошибки на обучающей выборке, умножаем его прогноз на скорость обучения (learning rate) и добавляем в общую композицию. Обычно градиентный бустинг продолжает до тех пор, пока не достигнута заданная максимальная глубина деревьев или пока не будет достигнуто заданное количество деревьев.

После того, как была построена модель градиентного бустинга, мы можем использовать ее для предсказания значений целевой переменной на новых объектах. Для этого на вход модели подается объект  $x$ , и вычисляется значение предсказания  $F(x)$ . Как и в случае обучения, вычисление предсказаний также осуществляется путем суммирования предсказаний всех деревьев.

Метод градиентного бустинга позволяет получить высокую точность предсказаний на различных задачах регрессии и классификации. Однако он может быть довольно требователен к вычислительным ресурсам и может иметь тенденцию к переобучению на небольших выборках. Для уменьшения риска переобучения можно использовать регуляризацию, например, ограничения на глубину деревьев или использование случайного выбора подмножества объектов и признаков на каждой итерации.

## Градиентный бустинг для классификации

Формальное определение градиентного бустинга для бинарной классификации можно записать следующим образом:

Пусть имеется обучающая выборка  $X = \{(x_1, y_1), \dots, (x_n, y_n)\}$ , где  $x_i \in \mathbb{R}^d$  - вектор признаков  $i$ -го объекта,  $y_i \in \{0, 1\}$  - метка класса объекта. Требуется построить функцию  $F(x)$ , которая будет предсказывать метку класса на новых объектах.

Аналогично регрессии, начинаем с построения начальной модели - константы  $F_0(x)$ . Затем на каждой следующей итерации  $m$  строится новое дерево решений  $h_m(x)$ , которое «исправляет» ошибки предыдущих деревьев. Функцию, которую мы пытаемся оптимизировать, можно определить как:

$$L(F) = \sum_{i=1}^n l(y_i, F(x_i))$$

где  $l(y, F(x))$  - функция потерь для объекта  $(x, y)$ , которая показывает, насколько сильно отличается предсказание  $F(x)$  от реальной метки  $y$ .

На каждой итерации мы хотим найти дерево  $h_m(x)$ , которое будет минимизировать функцию потерь  $L(F_{m-1} + h_m)$ :

$$h_m = \arg \min_h \sum_{i=1}^n l(y_i, F_{m-1}(x_i) + h(x_i))$$

Чтобы найти дерево  $h_m(x)$ , используется метод градиентного спуска, причем спуск производится не напрямую по функции  $L(F)$ , а с учетом ответов на текущей итерации. Градиент функции потерь для бинарной классификации выражается следующим образом:

$$\nabla L(y_i, F) = - \left( y_i - \frac{1}{1 + e^{-F(x_i)}} \right)$$

где  $F(x_i)$  - значение предсказания на  $i$ -м объекте на текущей итерации. Градиент функции потерь выражает, насколько сильно нужно изменить значение предсказания на данном объекте, чтобы уменьшить значение функции потерь.

После того, как найдено дерево  $h_m(x)$ , коэффициент  $\gamma_m$  определяется путем минимизации функции потерь по  $\gamma$ :

$$\gamma_m = \arg \min_\gamma \sum_{i=1}^n l(y_i, F_{m-1}(x_i) + \gamma h_m(x_i))$$

Итоговая функция предсказания  $F(x)$  на новом объекте вычисляется как сумма всех построенных деревьев с учетом их коэффициентов:

$$F(x) = \sum_{m=1}^M \gamma_m h_m(x)$$

где  $M$  - количество построенных деревьев.

На каждой итерации градиентного бустинга строится дерево, которое минимизирует функцию потерь, учитывая значения предсказания на предыдущих итерациях. Далее, коэффициенты для каждого дерева находятся путем решения задачи оптимизации. Итоговая функция предсказания получается как линейная комбинация всех деревьев с учетом их коэффициентов.

Для многоклассовой классификации градиентный бустинг обычно используется в сочетании с методом один против всех (one-vs-all), где для каждого класса строится своя модель градиентного бустинга, которая отличает этот класс от всех остальных.

## Подбор параметров для градиентного бустинга

Подбор оптимальных параметров для модели градиентного бустинга является важным этапом, который может существенно повлиять на качество предсказания.

Основными параметрами, которые нужно настроить, являются:

- Learning rate (шаг обучения) - коэффициент, на который умножается значение градиента на каждой итерации. Маленькие значения learning rate могут привести к более точным предсказаниям, но требуют большего числа итераций для обучения. Большие значения learning rate могут привести к более быстрой сходимости, но могут также привести к переобучению модели.
- Количество деревьев - определяет, сколько деревьев будет использоваться в модели. Большое количество деревьев может улучшить качество предсказания, но может также привести к переобучению модели.

- Максимальная глубина деревьев - определяет, сколько раз дерево будет делить данные. Большая глубина деревьев может привести к переобучению модели, тогда как слишком маленькая глубина может привести к недообучению.
- Минимальное число объектов в листе - определяет минимальное количество объектов, которые должны оказаться в листе дерева. Большее значение этого параметра может привести к сокращению переобучения модели.
- Размер выборки для построения деревьев - определяет, какое количество объектов будет выбрано для построения каждого дерева. Это может привести к уменьшению шума и улучшению качества предсказания.

Чтобы подобрать оптимальные параметры для модели градиентного бустинга, можно использовать перекрестную проверку (cross-validation), которая позволяет оценить качество модели на независимых выборках данных. Можно также использовать методы оптимизации, такие как Grid Search или Random Search, чтобы автоматически подобрать оптимальные значения параметров.

### Регуляризация для градиентного бустинга

Основные методы регуляризации для градиентного бустинга включают в себя:

- Ограничение глубины деревьев (`max_depth`) - установка максимальной глубины деревьев может предотвратить переобучение и ускорить обучение.
- Ограничение на число листьев (`max_leaf_nodes`) - установка максимального количества листьев может также предотвратить переобучение и упростить модель.
- Ограничение на минимальное количество выборок в листе (`min_samples_leaf`) - установка минимального количества выборок в листе может предотвратить переобучение, особенно если выборка маленькая.
- Ограничение на минимальное количество выборок в узле (`min_samples_split`) - установка минимального количества выборок в узле может предотвратить создание слишком мелких узлов, что может привести к переобучению.
- Ограничение на максимальное количество признаков при поиске наилучшего разделения (`max_features`) - установка максимального количества признаков при поиске наилучшего разделения может предотвратить переобучение и ускорить обучение.
- L1-регуляризация (`alpha`) добавляет штраф к абсолютному значению весов признаков. Это заставляет модель делать выбор между использованием всех признаков с низкой точностью или только нескольких признаков с высокой точностью.
- L2-регуляризация (`lambda`) добавляет штраф к квадрату весов признаков. Это заставляет модель предпочитать использование всех признаков с умеренной точностью, чем только нескольких признаков с высокой точностью.

Комбинация различных методов регуляризации может помочь создать более устойчивую модель, уменьшить переобучение и улучшить ее обобщающую способность.

### 6.1.3 Модификации градиентного бустинга

Существует множество модификаций градиентного бустинга, которые могут помочь улучшить его производительность и точность. Некоторые из них включают в себя:

- XGBoost - еще один быстрый и эффективный алгоритм градиентного бустинга, который использует решающие деревья и оптимизирует функцию потерь, добавляя регуляризацию и использование градиента второго порядка.
- LightGBM - быстрый и эффективный алгоритм градиентного бустинга, который использует специальные техники, такие как гистограммы признаков, для ускорения процесса обучения.
- CatBoost - алгоритм градиентного бустинга, который использует категориальные признаки, включая их в процесс обучения без необходимости предварительной обработки данных.

Основные отличия между XGBoost, LightGBM и CatBoost можно выделить следующим образом (Таблица 6.2):

Таблица 6.1: Сравнение основных свойств XGBoost, LightGBM и CatBoost

Свойство	XGBoost	LightGBM	CatBoost
Алгоритм	Градиентный бустинг	Градиентный бустинг	Градиентный бустинг
Поддержка GPU	Да	Да	Да
Распределенное обучение	Да	Да	Да
Поддержка категориальных признаков	Нет	Да	Да
Поддержка пропущенных значений	Да	Да	Да
Скорость обучения	Средняя	Высокая	Средняя
Размер модели	Средний	Маленький	Средний
Качество предсказаний	Высокое	Высокое	Высокое

Таблица 6.2: Сравнение основных свойств XGBoost, LightGBM и CatBoost

Здесь:

- Алгоритм указывает, что все три библиотеки реализуют градиентный бустинг.
- Поддержка GPU, распределенное обучение и поддержка пропущенных значений имеются во всех трех библиотеках.
- LightGBM и CatBoost поддерживают категориальные признаки, в то время как XGBoost этого не делает.
- Скорость обучения в LightGBM наиболее высокая, в XGBoost средняя, а в CatBoost - средняя.
- Размер модели в LightGBM наименьший, в XGBoost средний, а в CatBoost - средний.
- Качество предсказаний очень высокое во всех трех библиотеках, но в LightGBM и CatBoost оно наиболее высокое.

Более подробно о XGBoost, LightGBM и CatBoost мы поговорим далее.

#### 6.1.4 Вопросы для самопроверки

Какова сущность метода бустинга в машинном обучении?

1. Метод обучения без учителя
2. Метод генерации случайного леса
3. Метод последовательного построения композиции моделей
4. Метод независимого построения композиции моделей
5. Метод классификации объектов

Правильные ответы: 3

В чем суть алгоритма Adaboost (два ответа)?

1. AdaBoost обучает набор слабых моделей с помощью последовательно изменяющихся весов для объектов и комбинирует их в сильную модель.
2. AdaBoost для классификации выбирает на каждом шаге новую модель, которая за счет весов в большей степени сфокусирована на классификации неправильно классифицированных объектов в предыдущей модели.
3. AdaBoost использует градиентный спуск для обновления весов данных на каждом шаге обучения модели.
4. AdaBoost использует алгоритм случайного леса для комбинирования слабых моделей в сильную модель.

Правильные ответы: 1, 2

В чем суть алгоритма градиентного бустинга (два ответа)?

- Градиентный бустинг обучает набор деревьев решений на последовательно изменяющихся остатках и комбинирует их в сильную модель.
- Градиентный бустинг создает n независимых деревьев, на каждом из которых минимизирует функцию потерь в соответствии с градиентом.
- Градиентный бустинг использует градиентный спуск на остатках для оптимизации параметров моделей.

Правильные ответы: 1, 3

### 6.1.5 Резюме по разделу

Бустинг - это метод машинного обучения, который используется для улучшения качества прогнозов, создаваемых слабыми моделями машинного обучения. При бустинге каждая следующая модель (называемая базовой моделью) фокусируется на тех примерах данных, которые были неправильно классифицированы предыдущими моделями. В результате, каждая базовая модель «улучшает» работу предыдущих моделей, и в конечном итоге, бустинг представляет собой комбинацию всех базовых моделей, которые работают вместе, чтобы дать лучший результат, чем любая из них по отдельности.

Существует несколько подходов для реализации бустинга:

1. Adaboost
2. Градиентный бустинг

Adaboost - это алгоритм бустинга, который используется для улучшения качества классификации или регрессии в машинном обучении. Он работает путем создания последовательности слабых моделей машинного обучения и комбинирования их в единую сильную модель.

На каждой итерации Adaboost выбирает новую слабую модель, которая фокусируется на тех примерах данных, которые были классифицированы неправильно на предыдущих итерациях. Веса этих неправильно классифицированных примеров увеличиваются, чтобы следующая слабая модель сфокусировалась на них больше. Затем Adaboost комбинирует все слабые модели, чтобы получить сильную модель, которая дает более точные прогнозы.

Градиентный бустинг - это метод машинного обучения, который используется для создания сильной модели путем последовательного добавления слабых моделей, которые минимизируют ошибку предыдущих моделей.

В градиентном бустинге на каждой итерации создается новая слабая модель машинного обучения, которая пытается исправить ошибки предыдущей модели. Для этого градиентный бустинг использует градиент функции потерь, чтобы определить, какие примеры данных предыдущая модель классифицировала неправильно, и насколько сильно эти примеры влияют на функцию потерь.

Затем градиентный бустинг обучает новую слабую модель на этих неправильно классифицированных примерах и добавляет ее в сильную модель в виде слагаемого. Повторяя этот процесс много раз, градиентный бустинг создает последовательность слабых моделей, которые совместно дают более точные прогнозы.

## 6.2 XGBoost

**Цель занятия:** ученик может применить алгоритм XGBoost для решения задач классификации и регрессии на подготовленных и неподготовленных данных.

**План занятия:**

- Визуальная демонстрация алгоритма
- Подготовка данных для алгоритма
- Процесс обучения
- Оценка качества алгоритма
- Интерпретация признаков с помощью алгоритма
- Применение алгоритма
- Плюсы и минусы алгоритма
- Реализация алгоритма в Python

### 6.2.1 Визуальная демонстрация алгоритма

XGBoost (eXtreme Gradient Boosting) - это алгоритм машинного обучения, основанный на градиентном бустинге деревьев решений. Он был разработан в 2014 году и является одним из наиболее эффективных алгоритмов для задач классификации, регрессии и ранжирования.

XGBoost обучает ансамбль деревьев решений последовательно, приближаясь к оптимальному прогнозу с каждой новой итерацией. На каждой итерации алгоритм добавляет новое дерево, которое предсказывает остатки предыдущих деревьев. Таким образом, каждое новое дерево корректирует ошибки предыдущих деревьев.

По умолчанию XGBoost строит деревья level-wise, то есть строит все узлы на одном уровне перед переходом к следующему уровню (Рисунок 6.3). Однако существует также возможность

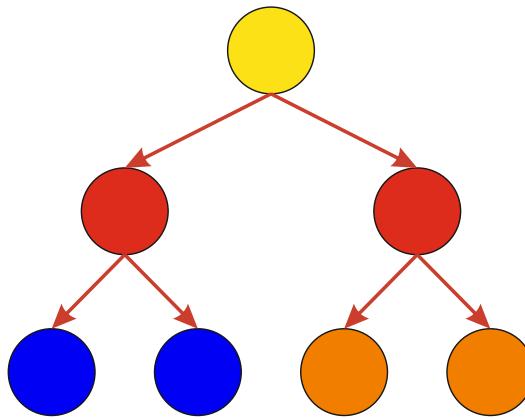


Рисунок 6.3: Level-wise построение дерева в XGBoost

строить деревья по листьям (leaf-wise), когда каждый узел строится таким образом, чтобы максимизировать уменьшение функции потерь. Это может привести к более быстрому обучению и более точным моделям, но может также вызывать переобучение в некоторых случаях. При выборе стратегии построения деревьев необходимо учитывать особенности конкретной задачи и набора данных.

Основные компоненты алгоритма XGBoost включают:

- Функцию потерь: Определяет, какой функционал оптимизируется в процессе обучения. В XGBoost используется функция потерь, которая состоит из двух частей: функции потерь для задачи (например, MSE для регрессии или логистическая функция потерь для классификации) и штрафа за сложность модели (регуляризация).
- Деревья решений: В качестве базовых алгоритмов используются решающие деревья. Каждое дерево строится на основе взвешенной версии обучающего набора данных, которая зависит от остатков предыдущих деревьев.
- Регуляризация: XGBoost поддерживает несколько методов регуляризации, включая L1 и L2 регуляризацию, ограничение глубины деревьев, ограничение на число листьев, ограничение на минимальное количество выборок в листе и ограничение на максимальное количество признаков при поиске наилучшего разделения. Регуляризация помогает предотвратить переобучение модели и повысить ее обобщающую способность.
- Градиентный спуск: Для нахождения оптимальных весов модели XGBoost использует градиентный спуск. На каждой итерации градиентного спуска алгоритм вычисляет градиент функции потерь по отношению к весам модели и изменяет их с определенным шагом в направлении уменьшения функции ошибки.

### 6.2.2 Подготовка данных для алгоритма

Шаги по предобработке наборов данных, полезные для использования с алгоритмом XGBoost. Некоторые из них включают:

- Обработка выбросов и пропущенных значений: XGBoost может работать с данными, содержащими пропущенные значения и выбросы, но для достижения лучшей производительности рекомендуется предварительно обработать эти аномалии.
- Обработка категориальных признаков: XGBoost поддерживает категориальные признаки, но они должны быть закодированы в числовом формате. Существуют различные методы кодирования категориальных признаков, такие как One-Hot Encoding, Label Encoding, и выбор метода зависит от конкретной задачи.
- Сбалансированность классов: Если ваш набор данных имеет несбалансированные классы, то может потребоваться принять меры для сбалансирования данных. Например, можно использовать взвешивание классов или сэмплирование данных, чтобы уравнять количество примеров в каждом классе.
- Отбор признаков: XGBoost может работать с большим количеством признаков, но для предотвращения переобучения модели может быть полезно отбирать только наиболее важные признаки. Это можно сделать с помощью методов отбора признаков, таких как Recursive Feature Elimination (RFE) или SelectKBest.

Это не полный список обработок данных, которые можно применить для использования с XGBoost, но это некоторые из наиболее распространенных методов. В целом, подготовка данных для использования с XGBoost зависит от конкретного набора данных и задачи машинного обучения.

### 6.2.3 Процесс обучения

Основная идея алгоритма XGBoost состоит в том, чтобы строить деревья решений последовательно, учитывая ошибки предыдущих деревьев, и объединять их в итоговую модель. Каждое новое дерево обучается на ошибках предыдущих деревьев, чтобы исправить

эти ошибки и улучшить качество модели. Этот процесс продолжается до тех пор, пока не будет достигнута определенная степень точности или не будут исчерпаны все ресурсы.

Основные шаги обучения алгоритма XGBoost:

1. Инициализация модели: Начальное значение целевой переменной устанавливается как среднее значение обучающей выборки.
2. Вычисление градиента и гессиана: Для каждого объекта в обучающей выборке вычисляются градиент и гессиан целевой функции. Градиент представляет собой первую производную функции ошибки, а гессиан - вторую производную (Градиент ошибки - это вектор первых частных производных функции ошибки по каждому параметру модели. Гессиан - это матрица вторых частных производных функции ошибки по каждой паре параметров модели). Градиент и гессиан используются для обучения каждого нового дерева решений.
3. Обучение дерева: Каждое новое дерево обучается на остатках (разнице между целевой переменной и предсказанным значением) предыдущих деревьев. Для этого используется метод градиентного бустинга, где дерево решений на каждой итерации обучается минимизировать целевую функцию, которая может быть выбрана из различных вариантов в зависимости от задачи.
4. Вычисление весов деревьев: Деревья решений имеют различные веса, которые определяют их важность в модели. Веса деревьев вычисляются на основе ошибок, которые они исправляют, и регуляризации, которая учитывает сложность модели.
5. Обновление целевой переменной: Целевая переменная обновляется путем вычитания предсказанного значения из текущего значения, что позволяет модели сфокусироваться на ошибках, которые еще не исправлены.
6. Повторение шагов: Шаги 2-6 повторяются до тех пор, пока не будет достигнута заданная точность или не будет исчерпано максимальное количество итераций.

Особенности обучения XGBoost:

- Обработка отсутствующих значений: XGBoost поддерживает обработку отсутствующих значений в данных, но лучше делать предобработку данных.
- Регуляризация: XGBoost использует регуляризацию для контроля сложности модели и избежания переобучения.
- Подбор гиперпараметров: XGBoost позволяет настраивать различные гиперпараметры, такие как глубина дерева, скорость обучения и количество деревьев. Это позволяет найти оптимальные настройки модели для каждой конкретной задачи.
- Автоматический выбор признаков: XGBoost может автоматически выбирать наиболее важные признаки для моделирования. Это происходит путем оценки важности признаков на основе их вклада в улучшение целевой функции.

Важные параметры при обучении XGBoost:

- learning\_rate: скорость обучения, которая определяет, насколько сильно корректируются веса при обновлении модели на каждой итерации. Выбор этого параметра влияет на скорость сходимости модели и на ее способность к обобщению.
- max\_depth: максимальная глубина дерева, которая определяет количество уровней дерева решений. Выбор этого параметра влияет на способность модели к обобщению и на скорость обучения.

- `min_child_weight`: минимальный вес дочернего узла, который определяет, какую минимальную величину должен иметь вес дочернего узла, чтобы продолжать делить узел. Выбор этого параметра влияет на устойчивость модели к шуму и на ее способность к обобщению.
- `gamma`: минимальное снижение функции ошибки, которое необходимо достичь, чтобы продолжать делить узел. Выбор этого параметра влияет на устойчивость модели к шуму и на скорость обучения.
- `subsample`: доля выборки, используемая для обучения каждого дерева. Выбор этого параметра влияет на способность модели к обобщению и на устойчивость к переобучению.
- `colsample_bytree`: доля признаков, используемых для обучения каждого дерева. Выбор этого параметра влияет на способность модели к обобщению и на устойчивость к переобучению.
- `alpha`: коэффициент L1-регуляризации весов деревьев. Выбор этого параметра влияет на скорость обучения и на способность модели к обобщению.
- `lambda`: коэффициент L2-регуляризации весов деревьев. Выбор этого параметра влияет на скорость обучения и на способность модели к обобщению.
- `num_boosted_round`: количество итераций обучения. Выбор этого параметра влияет на скорость обучения и на способность модели к обобщению.

Выбор оптимальных значений этих гиперпараметров может существенно повлиять на точность и обобщающую способность модели XGBoost.

#### 6.2.4 Оценка качества алгоритма

Для оценки качества алгоритма XGBoost часто используют следующие метрики:  
Классификация:

- **accuracy**:  $\frac{TP+TN}{TP+TN+FP+FN}$
- **precision**:  $\frac{TP}{TP+FP}$
- **recall**:  $\frac{TP}{TP+FN}$
- **F1**:  $2 * \frac{precision*recall}{precision+recall}$

Регрессия:

- **MSE**
- **MAE**

#### 6.2.5 Интерпретация признаков с помощью алгоритма

XGBoost имеет несколько методов для интерпретации важности признаков, которые могут помочь понять, какие признаки вносят наибольший вклад в модель. Рассмотрим несколько таких методов:

- Важность признаков на основе деревьев: этот метод основан на том, как деревья в модели XGBoost используют признаки для принятия решений. Для каждого признака суммируется значение важности, которое это свойство получает при разделении данных на каждом узле дерева. Чем выше суммарное значение важности, тем важнее признак.

- SHAP значения: SHAP (SHapley Additive exPlanations) - это метод, который предоставляет объяснение для каждого предсказания, показывая, насколько каждый признак вносит в конечный результат. Он использует теорию игр Шепли для распределения вклада между признаками.
- Partial Dependence Plot (PDP): PDP - это график, который показывает, как модель меняет свои прогнозы в зависимости от значений одного признака, при этом все остальные признаки остаются на своих местах. Это позволяет понять, какой эффект на прогноз оказывает каждый отдельный признак.
- Feature Interaction: XGBoost также предоставляет возможность выявления взаимодействия между признаками. Это может быть полезно, когда важность признаков взаимозависима, и один признак вносит больший вклад только вместе с другими признаками.

Все эти методы могут помочь понять, какие признаки вносят наибольший вклад в модель XGBoost и как они взаимодействуют между собой.

### 6.2.6 Применение алгоритма

XGBoost - это мощный алгоритм машинного обучения, который может применяться во многих областях. Вот несколько примеров его применения:

- Классификация: XGBoost может использоваться для классификации в различных областях, таких как медицина, финансы, обработка естественного языка и многих других.
- Регрессия: XGBoost может использоваться для решения задач регрессии, таких как прогнозирование цен на недвижимость, оценка риска финансовых инструментов и т.д.
- Ранжирование: XGBoost может использоваться для построения систем рекомендаций, ранжирования результатов поиска и т.д.
- Анализ данных: XGBoost может использоваться для поиска закономерностей и трендов в данных, а также для выявления аномалий и выбросов.
- Обработка изображений: XGBoost может использоваться для классификации изображений, определения объектов на изображении, обнаружения дефектов и т.д.
- Обработка звука: XGBoost может использоваться для классификации звуковых сигналов, распознавания речи, анализа звукового спектра и т.д.
- Промышленность: XGBoost может использоваться для оптимизации процессов в промышленности, таких как управление производственными линиями, оптимизация логистики и т.д.

В целом, XGBoost широко используется в различных областях, где требуется высокая точность и скорость работы алгоритма машинного обучения.

### 6.2.7 Плюсы и минусы алгоритма

#### Плюсы:

- Высокая производительность и масштабируемость на больших наборах данных.
- Превосходные результаты на различных задачах машинного обучения.
- Поддержка распределенных вычислений для обучения на кластерах.
- Реализация регуляризации для борьбы с переобучением.
- Возможность обработки разных типов данных (числовые, категориальные, текстовые и т.д.).

- Возможность обработки пропущенных значений.

**Минусы:**

- Большое количество гиперпараметров может быть сложным для настройки.
- Не так хорошо работает на маленьких выборках данных.
- Может потребоваться больше времени для обучения, чем для простых моделей машинного обучения, таких как линейные модели.

## 6.2.8 Реализация алгоритма в Python

XGBoost реализован в библиотеке xgboost (<https://xgboost.readthedocs.io/en/stable/>). Для задачи классификации с использованием XGBoost в библиотеке xgboost существует классификатор **XGBClassifier**, а для задач регрессии — **XGBRegressor**. Основные параметры этих моделей:

- **n\_estimators**: количество деревьев в лесу. По умолчанию `n_estimators=100`.
- **objective**: функция потерь, которую требуется минимизировать при обучении модели. Поддерживаются различные функции потерь в зависимости от задачи (например, «binary:logistic» для бинарной классификации или «reg:squarederror» для регрессии). По умолчанию `objective='reg:squarederror'`.
- **max\_depth**: максимальная глубина каждого дерева в лесу. По умолчанию `max_depth=6`.
- **learning\_rate**: шаг обучения (также называемый темпом обучения). По умолчанию `learning_rate=0.3`.
- **min\_child\_weight**: минимальный вес, необходимый для разделения узла. По умолчанию `min_child_weight=1`.
- **subsample**: доля обучающих данных, которые используются для каждого дерева. По умолчанию `subsample=1`.
- **colsample\_bytree**: доля признаков, которые используются для каждого дерева. По умолчанию `colsample_bytree=1`.
- **reg\_alpha**: коэффициент L1-регуляризации. По умолчанию `reg_alpha=0`.
- **reg\_lambda**: коэффициент L2-регуляризации. По умолчанию `reg_lambda=1`.

Классы XGBClassifier/XGBRegressor также имеют методы **fit(X, y)** для обучения модели на данных `X` и `y`, метод **predict(X)** для предсказания целевых значений для новых данных `X`, методы **score(X, y)** и **get\_params()** для получения оценки точности модели и параметров модели соответственно. Кроме того, класс XGBClassifier имеет метод **predict\_proba(X)**, который возвращает вероятности принадлежности каждому классу.

## 6.2.9 Вопросы для самопроверки

Как происходит построение дерева в XGBoost по умолчанию?

1. Level-wise
2. Leaf-wise
3. Oblivious decision tree

Правильные ответы: 1

Как XGBoost выбирает предикаты для разделения выборки?

1. XGBoost выбирает предикаты для разделения выборки случайным образом.
2. XGBoost выбирает предикаты для разделения выборки на основе их важности.
3. XGBoost выбирает предикаты для разделения выборки путем перебора всех возможных предикатов.
4. XGBoost выбирает предикаты для разделения выборки на основе Information Gain.

Правильные ответы: 4

Выберите плюсы алгоритма XGBoost:

1. Большое количество гиперпараметров может быть сложным для настройки.
2. Возможность обработки пропущенных значений.
3. Возможность обработки разных типов данных (числовые, категориальные, текстовые и т.д.).
4. Высокая производительность и масштабируемость на больших наборах данных.
5. Может потребоваться больше времени для обучения, чем для простых моделей машинного обучения, таких как линейные.
6. Не так хорошо работает на маленьких выборках данных.
7. Поддержка распределенных вычислений для обучения на кластерах.
8. Превосходные результаты на различных задачах машинного обучения.
9. Реализация регуляризации для борьбы с переобучением.

Правильные ответы: 2, 3, 4, 7, 8, 9

### 6.2.10 Резюме по разделу

XGBoost (Extreme Gradient Boosting) - это эффективный и мощный алгоритм машинного обучения, используемый для задач классификации и регрессии.

Основными преимуществами XGBoost являются высокая скорость обучения и предсказания, возможность работы с большими объемами данных, устойчивость к переобучению и возможность интерпретации результатов.

В XGBoost используется ансамбль деревьев решений, которые объединяются с помощью градиентного бустинга. Этот метод позволяет улучшать качество модели путем последовательного добавления новых деревьев и корректировки ошибок предыдущих.

XGBoost предлагает множество параметров для настройки модели, включая параметры дерева и параметры бустинга. Эти параметры позволяют контролировать глубину деревьев, скорость обучения, количество деревьев в ансамбле и другие параметры, влияющие на качество модели.

В целом, XGBoost является одним из наиболее эффективных и гибких инструментов машинного обучения, которые можно использовать для решения различных задач классификации и регрессии.

## 6.3 LightGBM

**Цель занятия:** ученик может применить алгоритм LightGBM для решения задач классификации и регрессии на подготовленных и неподготовленных данных.

**План занятия:**

- Визуальная демонстрация алгоритма
- Подготовка данных для алгоритма
- Процесс обучения
- Оценка качества алгоритма
- Интерпретация признаков с помощью алгоритма
- Применение алгоритма
- Плюсы и минусы алгоритма
- Реализация алгоритма в Python

### 6.3.1 Визуальная демонстрация алгоритма

LightGBM - это алгоритм градиентного бустинга деревьев решений, который разработан Microsoft и считается одним из наиболее быстрых и эффективных алгоритмов градиентного бустинга.

Основным преимуществом LightGBM является его способность работать с большими объемами данных и быстрая скорость обучения модели. Это достигается за счет использования нескольких техник оптимизации, таких как основанная на гистограммах обработка данных, локальной оценки градиента (GOSS) и сжатия данных (компактное представление данных). В LightGBM используется стратегия обучения по листьям (leaf-wise) в отличие от стратегии обучения по слоям (level-wise), используемой в других алгоритмах градиентного бустинга (Рисунок 6.4). Это означает, что при построении дерева на каждом шаге алгоритм выбира-

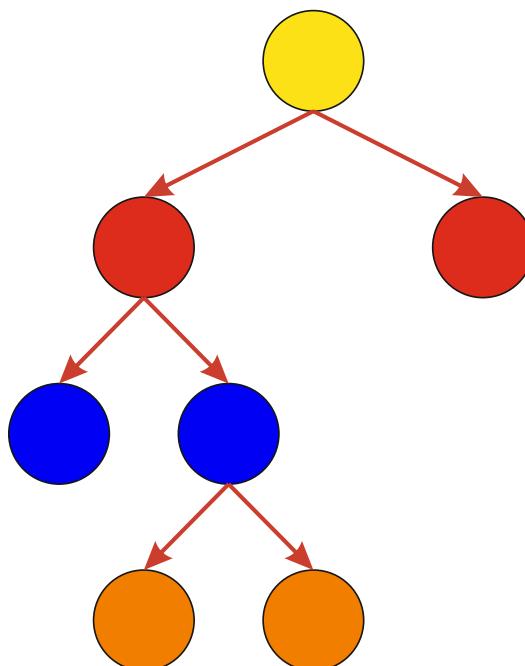


Рисунок 6.4: Leaf-wise построение дерева в LightGBM

ет тот лист, который максимально уменьшает функцию потерь, и строит дочерние узлы от этого листа. Таким образом, в LightGBM каждое дерево может иметь разную глубину, что позволяет модели улавливать более сложные зависимости в данных. Также это позволяет LightGBM строить деревья с меньшим количеством узлов, что уменьшает сложность модели и ускоряет время обучения. Однако, leaf-wise стратегия может привести к переобучению, особенно если выборка содержит выбросы или шумы. Поэтому, в LightGBM реализован ряд механизмов для предотвращения переобучения, таких как регуляризация и early stopping.

Кроме того, LightGBM поддерживает распределенное обучение на нескольких компьютерах, поддерживает многоклассовую классификацию и регрессию.

Общие шаги алгоритма LightGBM:

- Построение гистограмм для фичей
- Обучение деревьев решений с использованием градиентного бустинга
- Пересчет градиента и гессиана на каждой итерации для обновления весов деревьев
- Вычисление прогноза модели на новых данных

Таким образом, LightGBM - это быстрый и эффективный алгоритм градиентного бустинга, который может использоваться для решения задач классификации и регрессии на больших объемах данных.

### 6.3.2 Подготовка данных для алгоритма

Подготовка данных для алгоритма LightGBM может включать в себя следующие шаги:

- Обработка выбросов: для достижения лучшей производительности рекомендуется предварительно обработать выбросы.
- Обработка пропущенных значений: если есть пропущенные значения, заполните их средними значениями, медианами или другими подходящими значениями.

### 6.3.3 Процесс обучения

Процесс обучения LightGBM включает в себя следующие шаги:

1. Подготовка данных.
2. Подготовка параметров модели. LightGBM имеет множество гиперпараметров, которые могут быть настроены для достижения наилучшей производительности модели.
3. Создание деревьев. LightGBM использует алгоритм, основанный на градиентном бустинге деревьев решений. На каждой итерации модель добавляет новое дерево, которое пытается уменьшить остаточную ошибку предыдущей модели.
4. Прогнозирование. После построения модели можно использовать ее для прогнозирования новых значений. LightGBM позволяет делать прогнозы как для задач классификации, так и для задач регрессии.
5. Тюнинг гиперпараметров. Как и в случае с другими моделями машинного обучения, можно провести тюнинг гиперпараметров для достижения наилучшей производительности модели. LightGBM имеет множество гиперпараметров, которые могут быть настроены для улучшения качества модели.
6. Повторение. Шаги 2-4 могут быть повторены несколько раз для достижения наилучшей производительности модели.

В результате обучения LightGBM получается ансамбль деревьев решений, которые были построены поэтапно с помощью градиентного бустинга. LightGBM обладает высокой скоростью работы и хорошей производительностью на больших наборах данных. Кроме того, LightGBM поддерживает параллельное выполнение и распределенное обучение.

Особенности обучения LightGBM:

- Обработка отсутствующих значений: LightGBM также поддерживает обработку отсутствующих значений в данных, но лучше делать предобработку данных.
- Регуляризация: LightGBM использует регуляризацию для контроля сложности модели и избежания переобучения, но также использует методы сокращения данных и ограничения глубины деревьев.
- Подбор гиперпараметров: LightGBM позволяет настраивать множество гиперпараметров, таких как глубина дерева, скорость обучения, количество деревьев и другие параметры. Это позволяет найти оптимальные настройки модели для каждой конкретной задачи.
- Автоматический выбор признаков: LightGBM может автоматически выбирать наиболее важные признаки для моделирования. Это происходит путем оценки важности признаков на основе их вклада в улучшение целевой функции, но также можно использовать внешние алгоритмы выбора признаков.
- Параллельное обучение: LightGBM может обучаться параллельно на многих ядрах процессора, что позволяет сократить время обучения моделей и ускорить процесс выбора оптимальных гиперпараметров.

Важные параметры при обучении LightGBM:

- learning\_rate: скорость обучения, которая определяет, насколько сильно корректируются веса при обновлении модели на каждой итерации. Выбор этого параметра влияет на скорость сходимости модели и на ее способность к обобщению.
- max\_depth: максимальная глубина дерева, которая определяет количество уровней дерева решений. Выбор этого параметра влияет на способность модели к обобщению и на скорость обучения.
- min\_child\_samples: минимальное количество образцов, необходимых для создания нового узла. Выбор этого параметра влияет на устойчивость модели к шуму и на ее способность к обобщению.
- min\_child\_weight: минимальный вес дочернего узла, который определяет, какую минимальную величину должен иметь вес дочернего узла, чтобы продолжать делить узел. Выбор этого параметра влияет на устойчивость модели к шуму и на ее способность к обобщению.
- subsample: доля выборки, используемая для обучения каждого дерева. Выбор этого параметра влияет на способность модели к обобщению и на устойчивость к переобучению.
- colsample\_bytree: доля признаков, используемых для обучения каждого дерева. Выбор этого параметра влияет на способность модели к обобщению и на устойчивость к переобучению.
- reg\_alpha: коэффициент L1-регуляризации весов деревьев. Выбор этого параметра влияет на скорость обучения и на способность модели к обобщению.
- reg\_lambda: коэффициент L2-регуляризации весов деревьев. Выбор этого параметра влияет на скорость обучения и на способность модели к обобщению.

### 6.3.4 Оценка качества алгоритма

Для оценки качества алгоритма LightGBM часто используют следующие метрики:

Классификация:

- **accuracy:**  $\frac{TP+TN}{TP+TN+FP+FN}$
- **precision:**  $\frac{TP}{TP+FP}$
- **recall:**  $\frac{TP}{TP+FN}$
- **F1:**  $2 * \frac{precision*recall}{precision+recall}$

Регрессия:

- **MSE**
- **MAE**

### 6.3.5 Интерпретация признаков с помощью алгоритма

Чтобы интерпретировать признаки в LightGBM, можно использовать следующие методы:

- Важность признаков: LightGBM предоставляет встроенную функцию для оценки важности признаков. Эта функция рассчитывает важность признаков на основе их использования в деревьях решений. Важность признаков можно использовать для определения наиболее значимых признаков, влияющих на целевую переменную.
- SHAP значения: SHAP (SHapley Additive exPlanations) - это метод интерпретации модели, который позволяет определить влияние каждого признака на предсказание модели. LightGBM поддерживает SHAP значения, которые могут быть рассчитаны для каждого образца в данных. SHAP значения можно использовать для объяснения причин, по которым модель дает определенные предсказания.
- Визуализация деревьев решений: LightGBM позволяет визуализировать деревья решений, которые были созданы в ходе обучения модели. Визуализация деревьев позволяет легче понимать, какие признаки используются в модели и как они влияют на предсказание.
- Предсказания на новых данных: LightGBM позволяет сделать предсказания на новых данных. Если предсказания на новых данных достаточно точны, можно использовать эти предсказания для определения важности признаков. Если признаки не важны для предсказания на новых данных, то они могут быть удалены из модели.

### 6.3.6 Применение алгоритма

Основные области применения LightGBM включают в себя:

- **Финансы:** прогнозирование кредитных рисков, анализ финансовых рынков.
- **Реклама:** прогнозирование эффективности рекламы, оптимизация бюджета рекламных кампаний.
- **Интернет-магазины:** рекомендательные системы, прогнозирование продаж, сегментация аудитории.
- **Обработка естественного языка:** классификация текстов, анализ тональности, машинный перевод.

- **Изображения и видео:** распознавание объектов, классификация изображений, анализ видео.
- **Промышленность:** мониторинг и управление качеством продукции, прогнозирование отказов оборудования.
- **Транспорт и логистика:** прогнозирование спроса на перевозки, оптимизация маршрутов, анализ логистических данных.

В целом, LightGBM может быть полезным в любой области, где требуется классификация или регрессия на основе большого количества признаков и где есть достаточно данных для обучения модели. Он также может быть полезен для задач, связанных с обработкой естественного языка и изображений, и для задач прогнозирования в различных отраслях.

### 6.3.7 Плюсы и минусы алгоритма

#### Плюсы:

- Высокая скорость обучения: LightGBM использует алгоритм градиентного бустинга, который позволяет быстро обучать модель на больших объемах данных.
- Высокая точность предсказаний: LightGBM способен достичь высокой точности предсказаний, особенно при использовании большого количества деревьев.
- Эффективное использование памяти: LightGBM использует специальную структуру данных, которая позволяет эффективно использовать память и ускоряет обучение модели.
- Гибкость в настройке: LightGBM позволяет настраивать множество гиперпараметров, что позволяет добиться наилучшей производительности модели.

#### Минусы:

- Чувствительность к шуму и выбросам: LightGBM может быть чувствителен к шуму и выбросам в данных, что может приводить к переобучению модели.
- Неэффективность при наличии большого количества категориальных признаков с большим количеством уникальных значений: в этом случае может возникнуть проблема переобучения модели.
- Требуется подбор гиперпараметров: для достижения наилучшей производительности модели необходимо подобрать оптимальные гиперпараметры.
- Неинтерпретируемость: модель LightGBM не обеспечивает полного понимания, как именно происходит принятие решения.

### 6.3.8 Реализация алгоритма в Python

LightGBM реализован в библиотеке lightgbm (<https://lightgbm.readthedocs.io/en/latest/>). В библиотеке LightGBM для задач классификации существует класс **LGBMClassifier**, а для задач регрессии - класс **LGBMRegressor**. Основные параметры классов LGBMClassifier/LGBMRegressor:

- **objective:** целевая функция, которую оптимизирует LightGBM. Для задач классификации поддерживаются следующие целевые функции: «binary», «multiclass», «multiclassova», «cross\_entropy», «cross\_entropy\_lambda», «xentropy», «xentlambda», «lambdarank», «rank\_xendcg», «rank\_xendcg\_5», «rank\_xendcg\_10», «rank\_ndcg»,

«rank\_ndcg\_5», «rank\_ndcg\_10», «regression», «regression\_l1», «huber», «fair», «poisson», «gamma», «tweedie», «quantile», «mape», «gamma\_regression», «tweedie\_regression». Для задач регрессии поддерживаются следующие целевые функции: «regression», «regression\_l1», «huber», «fair», «poisson», «quantile», «mape», «gamma», «tweedie», «gamma\_regression», «tweedie\_regression».

- **n\_estimators:** количество деревьев в градиентном бустинге. По умолчанию n\_estimators=100.
- **learning\_rate:** шаг обучения, который контролирует вклад каждого дерева. По умолчанию learning\_rate=0.1.
- **max\_depth:** максимальная глубина каждого дерева. По умолчанию max\_depth=-1, что означает, что деревья разрастаются до тех пор, пока не будет достигнуто минимальное количество элементов для разделения.
- **num\_leaves:** максимальное количество листьев в каждом дереве. По умолчанию num\_leaves=31.
- **min\_data\_in\_leaf:** минимальное количество элементов, которые должны быть в листьях дерева. По умолчанию min\_data\_in\_leaf=20.
- **min\_child\_samples:** минимальное количество элементов, необходимое для того, чтобы узел мог быть разделен на два подузла. По умолчанию min\_child\_samples=20.
- **feature\_fraction:** количество признаков, которые должны быть рассмотрены при каждом разделении. По умолчанию feature\_fraction=1.0.
- **bagging\_fraction:** доля элементов, которые должны быть использованы при построении каждого дерева. По умолчанию bagging\_fraction=1.0.
- **bagging\_freq:** частота использования случайных элементов для построения каждого дерева. По умолчанию bagging\_freq=0.

Классы LGBMClassifier/LGBMRegressor имеют методы **fit(X, y)** для обучения модели на данных X и y, а также метод **predict(X)** для предсказания целевых значений для новых данных X. Кроме того, классы LGBMClassifier/LGBMRegressor имеют методы **score(X, y)** и **get\_params()** для получения оценки точности модели и параметров модели соответственно.

### 6.3.9 Вопросы для самопроверки

Как происходит построение дерева в LightGBM по умолчанию?

1. Level-wise
2. Leaf-wise
3. Oblivious decision tree

Правильные ответы: 2

Выберите плюсы алгоритма LightGBM:

1. Высокая скорость обучения: LightGBM использует алгоритм градиентного бустинга, который позволяет быстро обучать модель на больших объемах данных.
2. Высокая точность предсказаний: LightGBM способен достичь высокой точности предсказаний, особенно при использовании большого количества деревьев.
3. Гибкость в настройке: LightGBM позволяет настраивать множество гиперпараметров, что позволяет добиться наилучшей производительности модели.

4. Неинтерпретируемость: модель LightGBM не обеспечивает полного понимания, как именно происходит принятие решения.
5. Неэффективность при наличии большого количества категориальных признаков с большим количеством уникальных значений: в этом случае может возникнуть проблема переобучения модели.
6. Чувствительность к шуму и выбросам: LightGBM может быть чувствителен к шуму и выбросам в данных, что может приводить к переобучению модели.
7. Эффективное использование памяти: LightGBM использует специальную структуру данных, которая позволяет эффективно использовать память и ускоряет обучение модели.

Правильные ответы: 1, 2, 3, 7

### 6.3.10 Резюме по разделу

LightGBM - это быстрый и эффективный алгоритм градиентного бустинга деревьев решений, который использует уникальные техники построения деревьев и распределения данных по группам для достижения высокой скорости и точности предсказаний. LightGBM имеет ряд преимуществ перед другими алгоритмами градиентного бустинга, включая меньшее время обучения, более высокую скорость предсказания и более низкое потребление памяти. LightGBM поддерживает работу с категориальными признаками, автоматическую настройку гиперпараметров и распределенное обучение на нескольких процессорах.

## 6.4 Catboost

**Цель занятия:** ученик может применить алгоритм Catboost для решения задач классификации и регрессии на подготовленных и неподготовленных данных.

**План занятия:**

- Визуальная демонстрация алгоритма
- Подготовка данных для алгоритма
- Процесс обучения
- Оценка качества алгоритма
- Интерпретация признаков с помощью алгоритма
- Применение алгоритма
- Плюсы и минусы алгоритма
- Реализация алгоритма в Python

### 6.4.1 Визуальная демонстрация алгоритма

CatBoost - это алгоритм градиентного бустинга деревьев решений от компании Yandex. Он может обрабатывать большие объемы данных и имеет высокую скорость обучения модели, за счет использования нескольких техник оптимизации, таких как категориальная обработка данных, симметричный случайный выбор (Symmetric Random Selection) и использование градиентов второго порядка (Hessian).

Одним из основных преимуществ CatBoost является его способность автоматически обрабатывать категориальные признаки, не требуя их предварительной обработки, что может значительно ускорить процесс обучения модели.

В отличие от LightGBM, в CatBoost используется стратегия обучения по слоям (level-wise), которая состоит в том, что на каждом уровне дерева выбираются все узлы для разделения одновременно, что позволяет более полно использовать информацию о данных. Это может привести к более точной модели, но может также увеличить время обучения.

В Catboost используется структура дерева «Oblivious Decision Tree» - это алгоритм построения дерева решений, который относится к классу «неведущих» (oblivious) алгоритмов. В отличие от обычных деревьев решений, которые могут использовать различные признаки на каждом уровне дерева, неведущие деревья используют только один и тот же признак на каждом уровне дерева (Рисунок 6.5).

CatBoost поддерживает распределенное обучение на нескольких компьютерах, многоклассовую классификацию и регрессию.

Общие шаги алгоритма CatBoost:

1. Предобработка данных и подготовка набора признаков
2. Обучение базовых моделей на первоначальном наборе данных
3. Создание и обучение ансамбля базовых моделей с использованием градиентного бустинга
4. Вычисление ошибки на каждой итерации градиентного бустинга и пересчет весов моделей с учетом ошибки
5. Применение ансамбля моделей на новых данных для решения задачи классификации или регрессии

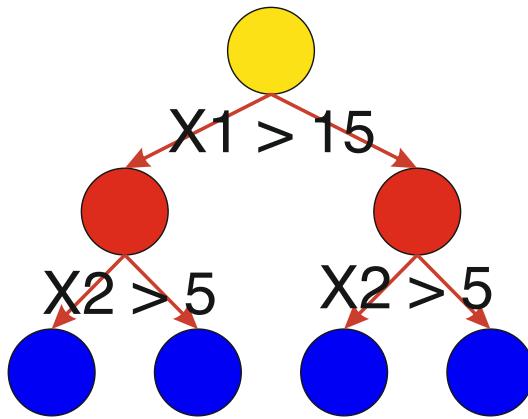


Рисунок 6.5: Oblivious Decision Tree в Catboost

CatBoost отличается от других алгоритмов градиентного бустинга тем, что он автоматически обрабатывает категориальные признаки и не требует их предварительного преобразования.

#### 6.4.2 Подготовка данных для алгоритма

Подготовка данных для алгоритма Catboost может включать в себя следующие шаги:

- Обработка пропущенных значений. Если есть пропущенные значения, заполните их средними значениями, медианами или другими подходящими значениями.

#### 6.4.3 Процесс обучения

Процесс обучения CatBoost включает в себя следующие шаги:

1. Подготовка данных и определение набора признаков.
2. Определение гиперпараметров модели, таких как количество деревьев, глубина деревьев, скорость обучения и т. д.
3. Обучение базовых моделей на первоначальном наборе данных.
4. Создание ансамбля базовых моделей с использованием градиентного бустинга.
5. Оценка качества модели на проверочном наборе данных для определения оптимального числа итераций и избежания переобучения.
6. Применение обученной модели для решения задачи классификации или регрессии.

CatBoost также поддерживает автоматическое кодирование категориальных признаков, встроенную кросс-валидацию и техники борьбы с переобучением, такие как регуляризация и сокращение градиента. Кроме того, CatBoost позволяет использовать несколько процессоров и графические процессоры для ускорения обучения модели.

Особенности обучения CatBoost:

- Обработка отсутствующих значений: CatBoost поддерживает обработку отсутствующих значений в данных, используя специальный токен для пропущенных значений, чтобы модель могла корректно обработать эти данные.

- Регуляризация: CatBoost использует регуляризацию для контроля сложности модели и избежания переобучения, но также использует методы сокращения данных и ограничения глубины деревьев.
- Подбор гиперпараметров: CatBoost позволяет настраивать множество гиперпараметров, таких как глубина дерева, скорость обучения, количество деревьев и другие параметры. Для нахождения оптимальных настроек модели можно использовать встроенные функции подбора гиперпараметров.
- Параллельное обучение: CatBoost также может обучаться параллельно на многих ядрах процессора, что позволяет сократить время обучения моделей и ускорить процесс выбора оптимальных гиперпараметров.
- Обработка категориальных признаков: CatBoost имеет уникальный алгоритм для работы с категориальными признаками, который позволяет автоматически преобразовывать категориальные признаки в числовые значения, не требуя предварительной обработки данных.

Важные параметры при обучении CatBoost:

- `learning_rate`: скорость обучения, которая определяет, насколько сильно корректируются веса при обновлении модели на каждой итерации. Этот параметр влияет на скорость сходимости модели и на ее способность к обобщению.
- `depth`: максимальная глубина дерева, которая определяет количество уровней дерева решений. Этот параметр влияет на способность модели к обобщению и на скорость обучения.
- `l2_leaf_reg`: коэффициент L2-регуляризации весов листьев деревьев. Этот параметр влияет на скорость обучения и на способность модели к обобщению.
- `min_data_in_leaf`: минимальное количество образцов в листе дерева. Этот параметр влияет на устойчивость модели к шуму и на ее способность к обобщению.
- `max_bin`: максимальное количество корзин для гистограммного метода построения деревьев. Этот параметр влияет на скорость обучения и на качество модели.
- `subsample`: доля выборки, используемая для обучения каждого дерева. Этот параметр влияет на способность модели к обобщению и на устойчивость к переобучению.
- `random_strength`: сила случайности, используемая при выборе случайных признаков для обучения каждого дерева. Этот параметр влияет на устойчивость к переобучению и на качество модели.
- `bagging_temperature`: температура распределения Больцмана, используемая при выборе объектов для обучения каждого дерева. Этот параметр влияет на способность модели к обобщению и на устойчивость к переобучению.

#### 6.4.4 Оценка качества алгоритма

Для оценки качества алгоритма Catboost часто используют следующие метрики:

Классификация:

- **accuracy**:  $\frac{TP+TN}{TP+TN+FP+FN}$
- **precision**:  $\frac{TP}{TP+FP}$
- **recall**:  $\frac{TP}{TP+FN}$
- **F1**:  $2 * \frac{precision*recall}{precision+recall}$

Регрессия:

- MSE
- MAE

#### 6.4.5 Интерпретация признаков с помощью алгоритма

CatBoost предлагает различные методы интерпретации признаков, которые могут помочь в понимании важности признаков и их влияния на предсказания модели:

- Важность признаков: CatBoost предоставляет встроенную функцию для оценки важности признаков. Эта функция рассчитывает важность признаков на основе их использования в деревьях решений. Важность признаков можно использовать для определения наиболее значимых признаков, влияющих на целевую переменную.
- SHAP значения: CatBoost также поддерживает расчет SHAP значений для интерпретации модели. SHAP значения позволяют определить влияние каждого признака на предсказание модели и объяснить, почему модель дает определенные предсказания. Это может быть полезным для понимания важности и влияния признаков.
- Визуализация деревьев: CatBoost позволяет визуализировать деревья решений, построенные в ходе обучения модели. Визуализация деревьев позволяет лучше понять, какие признаки используются в модели и как они влияют на предсказание.
- Предсказания на новых данных: При сделанных предсказаниях на новых данных в CatBoost можно оценить важность признаков. Если признаки не важны для предсказания на новых данных, то они могут быть удалены из модели.

Эти методы интерпретации признаков помогают разобраться в модели и понять, какие признаки наиболее важны для ее предсказаний.

#### 6.4.6 Применение алгоритма

CatBoost широко используется в различных областях и может быть применен для решения следующих задач:

- **Реклама и маркетинг:** прогнозирование кликов и конверсий, оптимизация рекламных кампаний, персонализация рекомендаций.
- **Финансы:** оценка кредитного scoringа, мошенническое обнаружение, прогнозирование рыночных трендов.
- **Здравоохранение:** диагностика заболеваний, прогнозирование рисков, анализ медицинских изображений.
- **Интернет-магазины:** рекомендательные системы, прогнозирование спроса, управление ассортиментом.
- **Телекоммуникации:** прогнозирование оттока клиентов, оптимизация сетей связи.
- **Транспорт и логистика:** маршрутизация, прогнозирование задержек, управление логистическими процессами.
- **Энергетика:** прогнозирование потребления энергии, оптимизация энергетических сетей.

CatBoost показывает хорошую производительность в задачах с большим числом категориальных признаков и может эффективно работать с разреженными данными. Он также может быть применен в задачах обработки естественного языка, компьютерного зрения и других областях, где требуется классификация и регрессия.

### 6.4.7 Плюсы и минусы алгоритма

#### Плюсы

- Обработка категориальных признаков: CatBoost автоматически обрабатывает категориальные признаки без необходимости их предварительного преобразования. Это упрощает процесс подготовки данных и позволяет моделировать задачи с большим количеством категориальных признаков.
- Устойчивость к переобучению: CatBoost включает в себя встроенные методы регуляризации, которые помогают предотвратить переобучение модели. Это делает его более устойчивым к шуму и выбросам в данных.
- Высокая точность и обобщающая способность: CatBoost показывает хорошие результаты на различных задачах и способен обобщать на новые данные. Он обладает высокой предсказательной точностью при правильной настройке гиперпараметров.
- Эффективное использование категориальных признаков: CatBoost использует специальный алгоритм для эффективной обработки категориальных признаков, что позволяет получать более точные предсказания.

#### Минусы

- Длительное время обучения: в некоторых случаях обучение CatBoost может требовать больше времени по сравнению с другими алгоритмами градиентного бустинга, особенно при большом количестве категориальных признаков.
- Требуется настройка гиперпараметров: CatBoost имеет множество гиперпараметров, которые необходимо настроить для достижения наилучшей производительности модели. Это может потребовать времени и вычислительных ресурсов.
- Высокое потребление памяти: CatBoost требует большого объема памяти для обработки больших объемов данных с большим количеством признаков, особенно при использовании GPU.

В целом, CatBoost является мощным алгоритмом градиентного бустинга с хорошей поддержкой категориальных признаков, но требует настройки гиперпараметров и может потребовать больше вычислительных ресурсов.

### 6.4.8 Реализация алгоритма в Python

CatBoost реализован в библиотеке `catboost` (<https://catboost.ai/en/docs/>). В библиотеке CatBoost для задач классификации существует класс **CatBoostClassifier**, а для задач регрессии - класс **CatBoostRegressor**. Основные параметры классов `CatBoostClassifier`/`CatBoostRegressor`:

- **loss\_function:** целевая функция, которую оптимизирует CatBoost. Для задач классификации поддерживаются следующие целевые функции: «Logloss», «CrossEntropy», «MultiClass», «MultiClassOneVsAll», «AUC», «F1», «Accuracy», «PRAUC», «Recall»,

«Precision», «BalancedAccuracy». Для задач регрессии поддерживаются следующие целевые функции: «MAE», «MAPE», «Poisson», «Quantile», «RMSE», «SquaredLoss», «Huber», «LogLinQuantile».

- **n\_estimators:** количество деревьев в градиентном бустинге. По умолчанию n\_estimators=1000.
- **learning\_rate:** шаг обучения, который контролирует вклад каждого дерева. По умолчанию learning\_rate=0.03.
- **max\_depth:** максимальная глубина каждого дерева. По умолчанию max\_depth=6.
- **num\_leaves:** максимальное количество листьев в каждом дереве. По умолчанию num\_leaves=31.
- **min\_data\_in\_leaf:** минимальное количество элементов, которые должны быть в листьях дерева. По умолчанию min\_data\_in\_leaf=1.
- **colsample\_bytree:** количество признаков, которые должны быть рассмотрены при каждом разделении на каждом уровне дерева. По умолчанию colsample\_bytree=1.0.
- **subsample:** доля элементов, которые должны быть использованы при построении каждого дерева. По умолчанию subsample=1.0.
- **random\_seed:** начальное значение для генератора случайных чисел. По умолчанию random\_seed=None.

Классы CatBoostClassifier/CatBoostRegressor имеют методы **fit(X, y)** для обучения модели на данных X и y, а также метод **predict(X)** для предсказания целевых значений для новых данных X. Кроме того, классы CatBoostClassifier/CatBoostRegressor имеют методы **score(X, y)** и **get\_params()** для получения оценки точности модели и параметров модели соответственно.

#### 6.4.9 Вопросы для самопроверки

Как происходит построение дерева в Catboost по умолчанию?

1. Level-wise
2. Leaf-wise
3. Oblivious decision tree

Правильные ответы: 3

Выберите плюсы алгоритма Catboost:

1. Высокая точность и обобщающая способность: CatBoost показывает хорошие результаты на различных задачах и способен обобщать на новые данные. Он обладает высокой предсказательной точностью при правильной настройке гиперпараметров.
2. Высокое потребление памяти: CatBoost требует большого объема памяти для обработки больших объемов данных с большим количеством признаков, особенно при использовании GPU.
3. Длительное время обучения: в некоторых случаях обучение CatBoost может требовать больше времени по сравнению с другими алгоритмами градиентного бустинга, особенно при большом количестве категориальных признаков.
4. Обработка категориальных признаков: CatBoost автоматически обрабатывает категориальные признаки без необходимости их предварительного преобразования. Это упрощает процесс подготовки данных и позволяет моделировать задачи с большим количеством категориальных признаков.

5. Требуется настройка гиперпараметров: CatBoost имеет множество гиперпараметров, которые необходимо настроить для достижения наилучшей производительности модели. Это может потребовать времени и вычислительных ресурсов.
6. Устойчивость к переобучению: CatBoost включает в себя встроенные методы регуляризации, которые помогают предотвратить переобучение модели. Это делает его более устойчивым к шуму и выбросам в данных.
7. Эффективное использование категориальных признаков: CatBoost использует специальный алгоритм для эффективной обработки категориальных признаков, что позволяет получать более точные предсказания.

Правильные ответы: 1, 4, 6, 7

#### 6.4.10 Резюме по разделу

CatBoost - это еще один быстрый и эффективный алгоритм градиентного бустинга деревьев решений, который также использует уникальные техники построения деревьев и обработки данных. Он также имеет ряд преимуществ перед другими алгоритмами, включая автоматическую обработку категориальных признаков, быструю скорость обучения и предсказания, а также поддержку распределенного обучения.

Одним из основных преимуществ CatBoost является его способность работать с категориальными признаками без необходимости их предварительной обработки, что может существенно упростить процесс подготовки данных. CatBoost также обеспечивает автоматическую настройку гиперпараметров, что может помочь улучшить качество модели. Кроме того, CatBoost может использоваться для задач классификации, регрессии и ранжирования.

## 6.5 Резюме по модулю

Бустинг является методом машинного обучения, который позволяет улучшить качество прогнозов, создаваемых слабыми моделями машинного обучения. При использовании этого метода каждая последующая базовая модель фокусируется на неправильно классифицированных примерах данных предыдущих моделей, что позволяет создать сильную модель, которая дает более точные прогнозы.

Существует несколько подходов для реализации бустинга, такие как Adaboost и градиентный бустинг. Adaboost работает путем создания последовательности слабых взвешенных моделей машинного обучения и комбинирования их в единую сильную модель. Градиентный бустинг использует градиент, настраиваемый на остатки (разность между предсказаниями текущей модели и правильными ответами предыдущих моделей), чтобы определить, какие примеры данных были неправильно классифицированы предыдущими моделями, и насколько сильно эти примеры влияют на функцию потерь.

XGBoost, LightGBM и CatBoost являются эффективными и мощными алгоритмами градиентного бустинга, используемыми для задач классификации и регрессии. Они используют градиентный бустинг деревьев решений для улучшения качества модели путем последовательного добавления новых деревьев и корректировки ошибок предыдущих.

Основными преимуществами XGBoost являются высокая скорость обучения и предсказания, возможность работы с большими объемами данных, устойчивость к переобучению и возможность интерпретации результатов. LightGBM имеет ряд преимуществ перед другими алгоритмами градиентного бустинга, включая меньшее время обучения, более высокую скорость предсказания и более низкое потребление памяти. CatBoost также имеет ряд преимуществ перед другими алгоритмами, включая автоматическую обработку категориальных признаков, быструю скорость обучения и предсказания, а также поддержку распределенного обучения.

В целом, все три алгоритма предоставляют множество параметров для настройки модели, влияющих на качество модели. Они также могут использоваться для задач классификации, регрессии и ранжирования. Однако, выбор конкретного алгоритма зависит от конкретных задач и требований к модели.

# Глава 7

## Обучение без учителя. Кластеризация. Снижение размерности данных.

### 7.1 Обучение без учителя

Обучение без учителя (англ. Unsupervised learning) - это подраздел машинного обучения, который занимается извлечением информации и паттернов из неструктурированных (неразмеченных) данных. В отличие от обучения с учителем, где для модели предоставляются размеченные данные с правильными ответами, при обучении без учителя данные не имеют меток или правильных ответов. Основная цель обучения без учителя - найти скрытую структуру в данных или сгенерировать полезные представления данных без явной разметки. В результате обучения без учителя модель выявляет внутренние закономерности, группирует или кластеризует данные, находит скрытые факторы или создает низкоразмерные представления данных.

Примеры алгоритмов обучения без учителя включают в себя методы кластеризации (например, k-средних, DBSCAN), методы снижения размерности (например, PCA, t-SNE), алгоритмы генеративных моделей (например, автокодировщики, генеративные состязательные сети) и многие другие.

Обучение без учителя широко применяется в анализе данных, обнаружении аномалий, сжатии данных, генерации контента и во многих других задачах, где необходимо извлекать полезную информацию из неразмеченных данных. Например, на рисунке показано совместное применение алгоритмов PCA (снижение размерности) + k-средних (кластеризация) для визуализации датасета MNIST (Рисунок 7.1). В итоге алгоритм находит центроиды кластеров<sup>2</sup> и закрашивает области, наиболее близкие к каждому центроиду, определенным цветом.

Давайте подробнее поговорим о двух классических подразделах обучения без учителя - кластеризации и снижении размерности данных.

#### 7.1.1 Кластеризация

Кластеризация в машинном обучении относится к задачам обучения без учителя и заключается в группировке объектов данных в кластеры на основе их схожести или близости друг

---

<sup>2</sup> Центроиды - это точки, представляющие центры каждого кластера, и они являются средними значениями точек внутри каждого кластера.

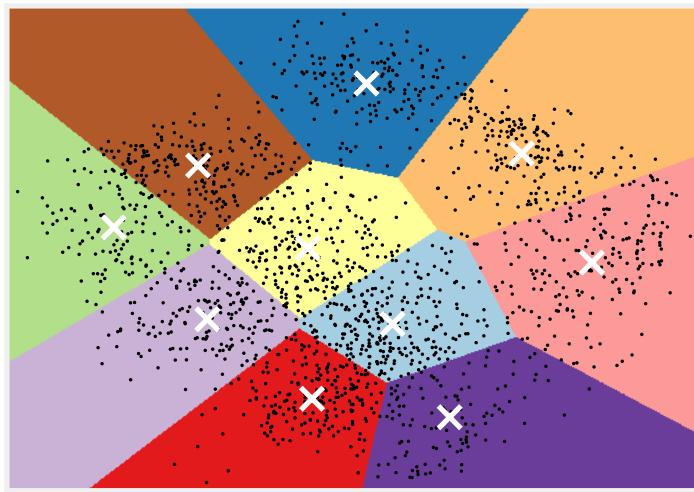


Рисунок 7.1: Визуализация датасета MNIST после применения алгоритмов снижения размерности и кластеризации

к другу. Алгоритмы кластеризации стремятся найти внутреннюю структуру в данных, основываясь на их признаках или свойствах, без заранее известного количества кластеров или информации о принадлежности объектов к определенным кластерам. Например, на рисунке 7.2 алгоритм Agglomerative Clustering выделяет монеты на фотографии. Целью кластеризации является максимизация схожести объектов внутри кластеров и минимизация схожести между кластерами.

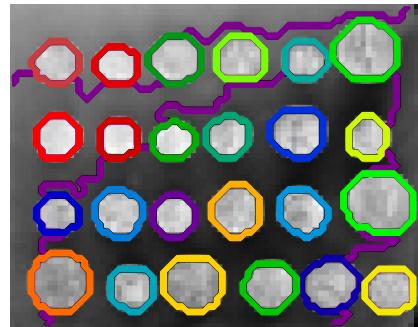
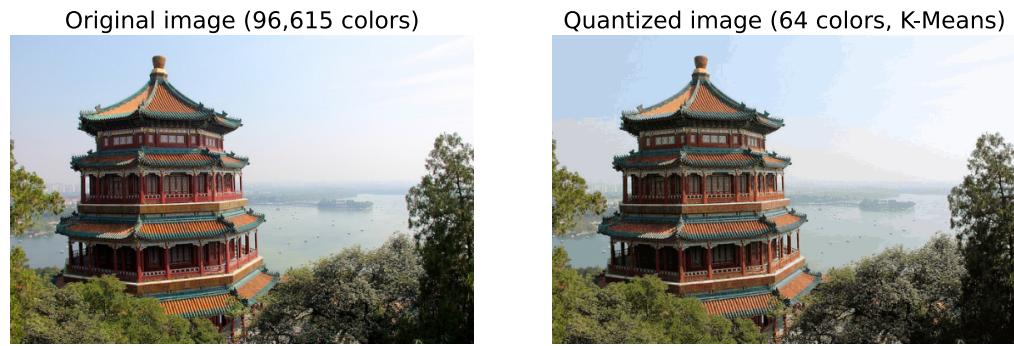


Рисунок 7.2: Кластеризация с помощью Agglomerative Clustering

На рисунке 7.3 показан пример сжатия данных с помощью алгоритма K-means. Распространенные примеры использования кластеризации:

- Поиск скрытых структур: Кластерный анализ помогает выявить скрытые структуры и группы в данных. Это может быть полезно для идентификации паттернов, тенденций



(1) Оригинальный рисунок с большим количеством цветов      (2) Обработанный рисунок с малым количеством цветов

Рисунок 7.3: Пример сжатия данных с помощью кластеризации

или сегментов, которые могут быть использованы для принятия более информированных решений.

- Сжатие данных: Путем группировки похожих объектов в кластеры можно сократить объем данных, сохраняя при этом основные характеристики и структуру. Это может упростить дальнейший анализ или обработку данных.
- Рекомендательные системы: Кластеризация может использоваться для создания групп пользователей или товаров с похожими характеристиками. Это может быть основой для разработки рекомендаций, чтобы предложить пользователям похожие товары или связать их с другими пользователями с общими интересами.
- Сегментация аудитории: Кластерный анализ может помочь в разделении аудитории на группы с общими характеристиками. Это может быть полезно для адаптации маркетинговых стратегий, персонализации коммуникации или создания целевых сообщений для каждой группы.

### Метрики качества кластеризации

Кластеры представляют собой группы объектов, которые обладают схожими характеристиками или свойствами, в то время как объекты из разных кластеров сильно отличаются друг от друга. Для измерения расстояний вводится понятие центроида. Внутрикластерное расстояние и межкластерное расстояние - это две основные характеристики, которые учитываются при оценке качества кластеризации:

- Внутрикластерное расстояние (интракластерное расстояние) - это мера сходства между объектами внутри одного кластера. Чем меньше среднее внутрикластерное расстояние, тем более компактными и однородными являются кластеры (Формула 7.1).

$$\text{Среднее внутрикластерное расстояние} = \frac{1}{N} \sum_{i=1}^N \sum_{\substack{j=1 \\ j \neq i}}^N \text{расстояние}(x_i, x_j) \quad (7.1)$$

где  $N$  - общее количество объектов,  $x_i$  и  $x_j$  - объекты, а  $\text{расстояние}(x_i, x_j)$  - функция, вычисляющая расстояние между объектами  $x_i$  и  $x_j$ . Обычно в качестве меры расстояния

используют L2 норму. Эта формула вычисляет сумму всех парных расстояний между объектами внутри каждого кластера и затем усредняет это значение по всем кластерам. Низкое внутрикластерное расстояние указывает на то, что объекты внутри кластера находятся близко друг к другу, что является желательным свойством для качественной кластеризации. Определение низкого внутрикластерного расстояния зависит от контекста и специфики данных. Универсального значения, которое можно было бы считать низким для всех случаев, нет.

- Межкластерное расстояние - это мера различия между кластерами. Оно определяет, насколько различные кластеры удалены друг от друга. Чем больше межкластерное расстояние, тем более разделены и отделены друг от друга кластеры. Высокое межкластерное расстояние указывает на хорошую разделимость и различимость кластеров. Среднее межкластерное расстояние может быть представлено следующим образом (Формула 7.2):

$$\text{Среднее межкластерное расстояние} = \frac{1}{K(K-1)} \sum_{i=1}^K \sum_{j=1, j \neq i}^K \text{расстояние}(c_i, c_j) \quad (7.2)$$

где  $K$  - общее количество кластеров,  $c_i$  и  $c_j$  - центроиды (средние значения) кластеров, а  $\text{расстояние}(c_i, c_j)$  - функция, вычисляющая расстояние между центроидами  $c_i$  и  $c_j$ . Обычно в качестве меры расстояния используют L2 норму. Эта формула вычисляет сумму всех парных расстояний между центроидами кластеров и затем усредняет это значение по всем парам кластеров.

Вот несколько основных метрик качества кластеризации, построенных с помощью учета расстояния:

- Коэффициент силуэта (англ. Silhouette Coefficient) (Формула 7.3):

$$SC = \frac{M - S}{\max(S, M)} \quad (7.3)$$

где  $M$  - среднее межкластерное расстояние,  $S$  - среднее внутрикластерное расстояние. Коэффициент силуэта принимает значения от -1 до 1, где ближе к 1 указывает на хорошую кластеризацию, а ближе к -1 - на неправильную кластеризацию. Формула называется «коэффициентом силуэта» (Silhouette Coefficient) потому, что она основана на понятии «силуэта» как визуального представления объекта или кластера.

Силуэт представляет собой меру компактности и разделенности объекта в контексте кластеризации. Визуально, силуэт объекта можно представить как его очертание, отражающее степень близости объекта к другим объектам внутри его кластера и удаленности от объектов в других кластерах. Она отражает, насколько хорошо объекты сгруппированы внутри своих кластеров, и насколько различаются между собой разные кластеры.

- Индекс Данна (англ. Dunn Index) (Формула 7.4):

$$D = \frac{\min_{1 \leq i \leq k, 1 \leq j \leq k, i \neq j} M(i, j)}{\max_{1 \leq l \leq k} d(l)} \quad (7.4)$$

где  $M(i, j)$  - расстояние между кластерами  $C_i$  и  $C_j$ ,  $k$  - общее количество кластеров, которые были образованы в результате кластеризации данных, а  $d(l)$  - диаметр кластера

$C_l$ . Чем больше значение индекса Данна, тем лучше разделены кластеры. Это означает, что межкластерные расстояния максимизированы, а внутрикластерные расстояния минимизированы. Более четкое разделение кластеров приводит к более высокому значению индекса Данна. В отличие от коэффициента силуэта, индекс Данна не имеет фиксированного диапазона значений, и его интерпретация может быть более сложной. Обычно сравниваются несколько различных кластерных разбиений, и кластеризация с более высоким значением индекса Данна считается более оптимальной.

- Индекс Дэвиса-Боулдина (англ. Davies-Bouldin Index) (Формула 7.5):

$$DB = \frac{1}{k} \sum_{i=1}^k \max_{j \neq i} \left( \frac{S_i + S_j}{M_{ij}} \right) \quad (7.5)$$

где  $S_i$  - среднее расстояние между объектами внутри кластера  $C_i$ ,  $M_{ij}$  - расстояние между центроидами кластеров  $C_i$  и  $C_j$ ,  $k$  - общее количество кластеров, которые были образованы в результате кластеризации данных. Чем меньше значение индекса Дэвиса-Боулдина, тем лучше разделены кластеры. Это означает, что внутрикластерные схожести максимизированы, а межкластерные различия минимизированы. Более четкое разделение кластеров приводит к более низкому значению индекса Дэвиса-Боулдина.

Сложность кластеризации заключается в том, что на одной выборке может быть несколько различных вариантов разделения на кластеры, и не всегда ясно, какое разбиение является правильным. Формализация критериев для определения правильного разбиения на практике достаточно сложна, поэтому сама задача кластеризации не всегда имеет однозначное решение.

В отсутствии разметки и при неизвестном заранее количестве кластеров, на практике одной из лучших метрик является коэффициент силуэта. Он позволяет оценить качество кластеризации, принимая во внимание меру компактности кластеров и их отделенность друг от друга. Коэффициент силуэта позволяет выбирать наилучший вариант кластеризации в таких случаях.

## Распространенные алгоритмы кластеризации

Существует множество алгоритмов кластеризации, каждый из которых имеет свои особенности и применимость в различных ситуациях. Вот наиболее распространенные:

- К-средних (K-means): Он разбивает данные на  $K$  кластеров, где  $K$  задается пользователем. Алгоритм итеративно оптимизирует положения центроидов кластеров и присваивает объекты к ближайшим центроидам. Это один из наиболее широко используемых алгоритмов кластеризации.
- DBSCAN (Density-Based Spatial Clustering of Applications with Noise): Этот алгоритм основан на плотности данных. Он ищет плотные области в пространстве данных и формирует кластеры на основе связанных плотных областей. DBSCAN может обнаруживать кластеры произвольной формы и обрабатывать выбросы.
- Иерархическая кластеризация: Это семейство алгоритмов, которые строят иерархическую структуру кластеров. Два основных подхода в иерархической кластеризации: агломеративный и дивизионный. Агломеративные методы действуют от частного к общему: начинают с отдельных объектов и последовательно объединяют их в кластеры, пока не будет достигнуто заданное условие остановки. Дивизионные методы действуют от

общего к частному: начинают с одного крупного кластера и разделяют его на более мелкие, пока не будут получены отдельные кластеры.

- Агломеративная кластеризация (Agglomerative Hierarchical Clustering): Это вариант иерархической кластеризации, где начинают с отдельных объектов и последовательно объединяют их на основе среднего расстояния между кластерами. Алгоритм объединяет кластеры, пока не будет достигнуто заданное количество кластеров.
- GMM (Gaussian Mixture Model): Этот алгоритм моделирует данные с использованием смеси нормальных (гауссовых) распределений. Каждый компонент смеси соответствует одному кластеру, и модель пытается найти наиболее вероятную смесь, описывающую данные.

Ниже представлено сравнение приведенных алгоритмов в форме таблицы (Таблица 7.1):

Метод	Основа алгоритма	Входные данные	Фиксированное количество кластеров	Форма кластеров	Обнаружение выбросов
К-средних	Расстояние между объектами и центроидами	Фактические наблюдения	Да	Сферические кластеры	Нет
DBSCAN	Плотность регионов в данных	Фактические наблюдения или попарные расстояния между наблюдениями	Нет	Произвольная	Да
Иерархическая кластеризация	Расстояние между объектами	Попарные расстояния между наблюдениями	Нет	Произвольная	Нет
Агломеративная кластеризация	Расстояние между объектами	Попарные расстояния между наблюдениями	Нет	Произвольная	Нет
GMM (Gaussian Mixture Model)	Смесь гауссовых распределений	Фактические наблюдения	Да	Сферические кластеры с различными ковариационными структурами	Да

Таблица 7.1: Сравнение основных алгоритмов кластеризации

В этой таблице представлены основные характеристики каждого метода, их применимость в различных ситуациях и преимущества. Выбор алгоритма зависит от характеристик данных, размерности пространства, предполагаемого числа кластеров и других факторов.

Также приведем визуальное сравнение работы алгоритмов кластеризации с сайта sklearn (Рисунок 7.4).

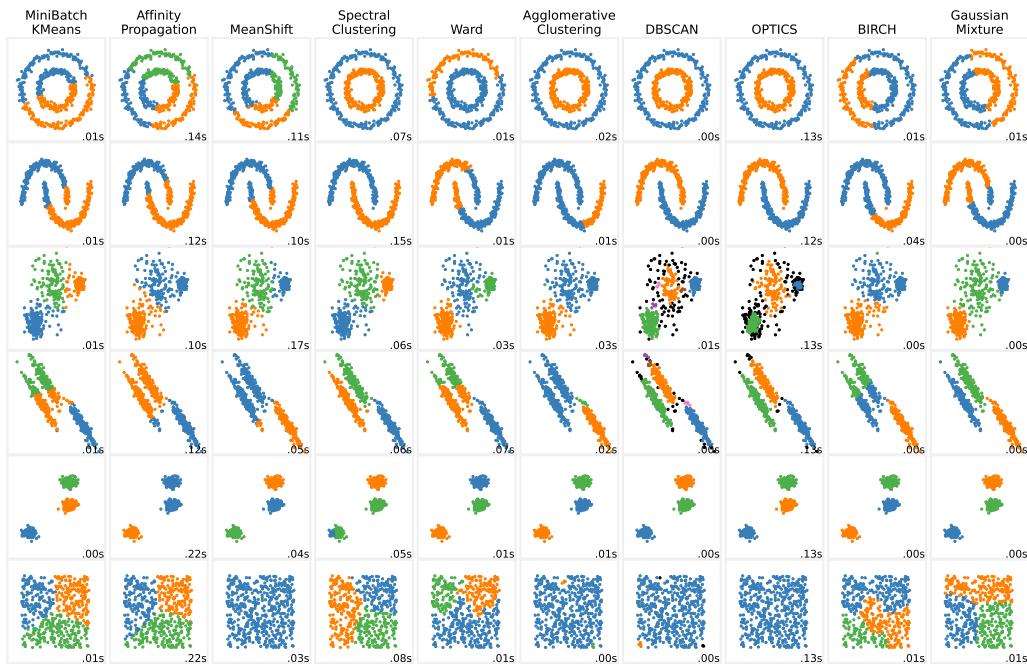


Рисунок 7.4: Сравнение качества работы алгоритмов кластеризации с помощью графиков

На рисунке 7.4 представлена работа алгоритмов с разными датасетами: вложенные окружности, наборы данных в виде дуг (подков), три близко друг к другу расположенных кластера, три параллельных кластера, три отдаленных друг от друга кластера, один кластер. Качество работы алгоритма измеряется тем, насколько корректно он кластеризует эти данные и тем, сколько времени он потратит на это (информация о скорости работы алгоритма представлена в левом нижнем углу). Графическое сравнение алгоритмов на рисунке 7.4 показывает, что наиболее качественные результат кластеризации показывают DBSCAN и OPTICS, чуть менее качественные - агломеративная кластеризация. Наиболее сложные датасеты (строка один, два и шесть) кластеризуются лучше с помощью DBSCAN и OPTICS, однако DBSCAN более оптимален с точки зрения скорости работы.

Алгоритмы K-means, DBSCAN и агломеративная кластеризация далее будут рассмотрены более подробно.

## 7.1.2 Снижение размерности данных

Снижение размерности данных - это процесс уменьшения числа признаков (измерений) в наборе данных. Оно выполняется с целью упрощения анализа данных, устранения шума, избавления от избыточной информации или подготовки данных для последующего использования в задачах машинного обучения.

Алгоритмы снижения размерности данных применяются в различных задачах анализа данных и машинного обучения. Вот несколько примеров задач, для которых широко используются методы снижения размерности:

- Визуализация данных: Снижение размерности помогает визуализировать данные высокой размерности в двух или трех измерениях для лучшего понимания структуры данных, обнаружения паттернов или визуального анализа.
- Отбор признаков: Методы снижения размерности могут использоваться для извлечения наиболее информативных признаков из исходных данных. Снижение размерности может быть применено перед построением моделей машинного обучения, чтобы уменьшить размерность входных данных и удалить избыточные или неинформативные признаки. Это может улучшить производительность моделей и сократить вычислительную сложность.
- Устранение шума: Методы снижения размерности могут помочь устранить шум в данных, фильтруя или игнорируя менее значимые компоненты.
- Сжатие данных для уменьшения расхода памяти.

Это некоторые типы задач, для которых применяются алгоритмы снижения размерности данных. Выбор конкретного метода зависит от конкретной задачи, типа данных и требований анализа.

## Распространенные алгоритмы снижения размерности данных

Существует множество методов для снижения размерности данных. Рассмотрим некоторые из них:

- Метод главных компонент (PCA): Он выполняет линейное преобразование данных, чтобы получить новые некоррелированные переменные, называемые главными компонентами. PCA — наиболее популярный метод снижения размерности.
- Метод t-SNE (t-Distributed Stochastic Neighbor Embedding): t-SNE позволяет визуализировать данные высокой размерности, сохраняя относительные расстояния между объектами. Он стремится сохранить соседство объектов в исходном пространстве и проектировать их на низкоразмерное пространство для визуализации.
- Метод линейного дискриминантного анализа (англ. Linear Discriminant Analysis, LDA): Линейный дискриминантный анализ - это статистический метод, используемый для анализа и классификации данных. Он относится к области обучения с учителем, где каждому образцу данных присваивается определенная метка класса. Основная цель LDA состоит в том, чтобы найти линейные комбинации признаков, которые наилучшим образом разделяют различные классы данных. Это достигается путем максимизации разделения между классами и минимизации разброса внутри каждого класса.
- Автоэнкодеры (Autoencoders): Автоэнкодеры являются нейронными сетями, которые обучаются восстанавливать входные данные на выходе. Внутри модели создаются представления данных меньшей размерности, которые модель затем пробует восстановить до исходной размерности. Автоэнкодеры могут быть использованы для снижения размерности и извлечения наиболее информативных признаков из данных.
- Корреляционный анализ: Корреляционный анализ - это статистический метод, который используется для измерения степени связи между двумя или более переменными. Он позволяет определить, насколько тесно связаны две переменные и какие типы связей между ними существуют. В корреляционном анализе используется коэффициент корреляции, который измеряет степень линейной зависимости между переменными.

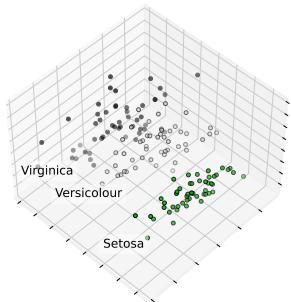
- Информативный признаковый отбор (Information Gain) в алгоритмах на деревьях: Информативный признаковый отбор, или Information Gain, является методом выбора наиболее информативных признаков при построении деревьев решений или других алгоритмов, основанных на деревьях, таких как случайный лес.
- Алгоритм регуляризации L1: Применение L1 регуляризации приводит к решению, при котором некоторые коэффициенты модели становятся равными нулю. Это позволяет выполнить отбор признаков, исключая несущественные признаки из модели. Признаки с нулевыми коэффициентами считаются неинформативными или слабо влияющими на целевую переменную.

Ниже представлено сравнение приведенных алгоритмов в форме таблицы (Таблица 7.2):

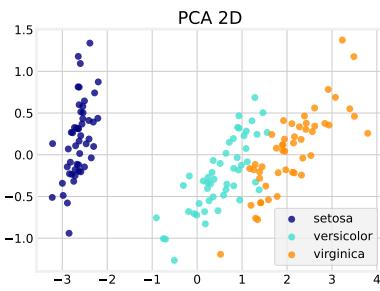
Метод	Описание	Преимущества	Недостатки
PCA	находит новое пространство признаков, состоящее из главных компонент исходных данных.	Простота и высокая скорость вычислений	Потеря информации о классификации
t-SNE	строит карту точек данных в низкоразмерном пространстве, сохраняя локальную структуру.	Сохранение локальной структуры данных	Вычислительно сложен и может быть медленным для больших наборов данных
LDA	находит новое пространство признаков, максимизируя разделение классов данных.	Учет информации о классификации	Могут возникнуть проблемы с переобучением, когда классов данных много или данные несбалансированы
Autoencoders	нейронные сети, которые обучаются восстанавливать входные данные с помощью сжатого представления.	Гибкость в моделировании сложных нелинейных отображений данных	Обучение требует больше вычислительных ресурсов и времени в сравнении с другими методами снижения размерности. Могут возникнуть проблемы с переобучением.

Таблица 7.2: Сравнение основных алгоритмов снижения размерности данных

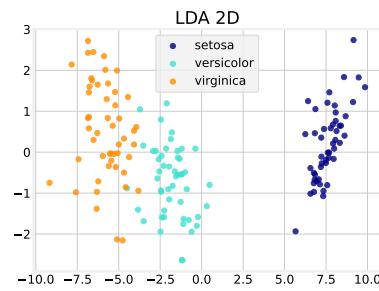
В данной таблице представлены основные характеристики методов снижения размерности данных: PCA, t-SNE, LDA и Autoencoders. Каждый метод обладает своими особенностями, применимостью и преимуществами. Это позволяет выбрать подходящий в зависимости от целей и требований конкретной задачи снижения размерности данных. Пример визуализации датасета Iris с помощью алгоритмов снижения размерности PCA и LDA представлен на рисунке 7.5.



(1) Данные Iris, 3 признака



(2) Визуализация Iris с PCA



(3) Визуализация Iris с LDA

Рисунок 7.5: Визуализации датасета Iris с помощью алгоритмов снижения размерности PCA и LDA

Алгоритмы PCA и t-SNE далее будут рассмотрены более подробно.

### 7.1.3 Вопросы для самопроверки

Для чего могут использоваться алгоритмы кластеризации данных (три ответа)?

1. Поиск скрытых структур
2. Задача регрессии
3. Сжатие данных
4. Рекомендательные системы

Правильные ответы: 1, 3, 4

Какие из приведенных алгоритмов относятся к алгоритмам кластеризации (два ответа)?

1. kNN
2. K-средних
3. DBSCAN
4. AdaBoost

Правильные ответы: 2, 3

Какие из приведенных алгоритмов относятся к алгоритмам снижения размерности (два ответа)?

1. GMM
2. t-SNE
3. DBSCAN
4. PCA

Правильные ответы: 2, 4

### 7.1.4 Резюме по разделу

Обучение без учителя (англ. Unsupervised Learning) - это раздел машинного обучения, в котором модель обучается на неразмеченных данных, то есть данных, не имеющих заранее

определенных меток или целевых переменных. В отличие от обучения с учителем, где модель обучается на размеченных данных с известными метками классов или целевыми значениями, обучение без учителя направлено на обнаружение скрытых структур, закономерностей и интересных паттернов в данных.

Кластеризация и снижение размерности данных являются классическими подразделами обучения без учителя.

1. Кластеризация:

- Кластеризация является методом группировки данных в кластеры с похожими объектами.
- Распространенные метрики качества для оценки эффективности кластеризации: среднее внутрикластерное расстояние, среднее межкластерное расстояние, коэффициент силуэта, индекс Данна, индекс Дэвиса-Боулдина
- Распространенные алгоритмы кластеризации включают K-средних, DBSCAN, иерархическую кластеризацию и GMM.

2. Снижение размерности данных:

- Снижение размерности данных - это процесс уменьшения количества признаков в данных.
- Распространенные алгоритмы снижения размерности данных включают PCA (метод главных компонент), t-SNE (t-distributed stochastic neighbor embedding), LDA (линейный дискриминантный анализ) и автокодировщики (autoencoders).

## 7.2 K-means

**Цель занятия:** ученик может применить алгоритм K-means для решения задач кластеризации на подготовленных и неподготовленных данных.

**План занятия:**

- Визуальная демонстрация алгоритма
- Подготовка данных для алгоритма
- Процесс обучения
- Оценка качества алгоритма
- Применение алгоритма
- Плюсы и минусы алгоритма
- Реализация алгоритма в Python

### 7.2.1 Визуальная демонстрация алгоритма

K-средних (k-means) - это один из наиболее распространенных методов кластеризации, используемых в машинном обучении и анализе данных. Он представляет собой алгоритм, который разбивает набор данных на заранее заданное количество кластеров, где каждый кластер представляет группу объектов, которые схожи между собой и отличаются от объектов в других кластерах (Рисунок 7.6).

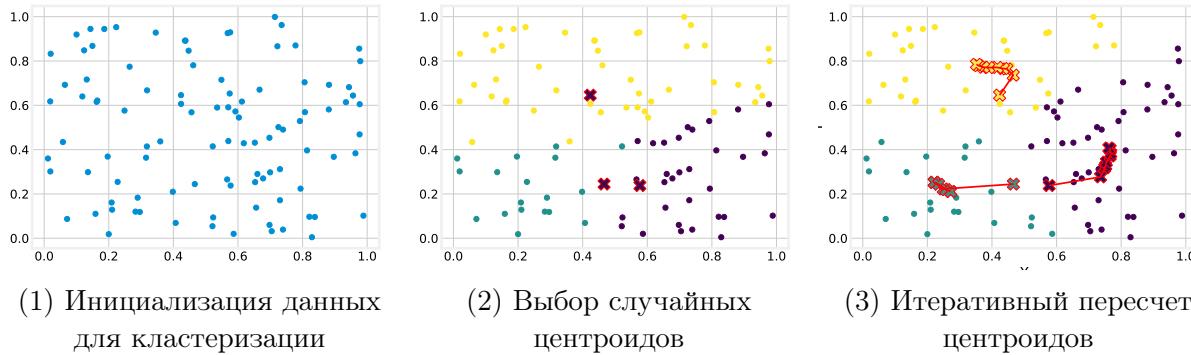


Рисунок 7.6: Процесс обучения K-means

На начальном этапе мы выбираем случайные центры для каждого кластера. Затем мы относим каждую точку к ближайшему кластеру на основе расстояния до центров. После этого мы пересчитываем центры кластеров, используя среднее арифметическое точек, принадлежащих кластеру.

Теперь, когда центры кластеров обновлены, мы повторяем процесс перераспределения точек по кластерам и обновления центров. Этот итеративный процесс повторяется до тех пор, пока центры кластеров и распределение точек не стабилизируются или достигнутся заранее заданные условия остановки.

Алгоритм k-means пытается минимизировать сумму квадратов расстояний между точками и центрами кластеров, что приводит к тому, что точки внутри каждого кластера становятся более близкими друг к другу, а точки между разными кластерами остаются далекими друг от друга.

### 7.2.2 Подготовка данных для алгоритма

Подготовка данных для алгоритма К-средних включает несколько шагов. Вот общий процесс подготовки данных для использования с алгоритмом К-средних:

- Масштабирование данных: Рекомендуется масштабировать признаки входных данных перед применением К-средних, особенно если признаки имеют различные шкалы или единицы измерения. Обычно используется стандартизация или нормализация данных для приведения их к общей шкале.
- Удаление выбросов: Выбросы могут негативно влиять на процесс кластеризации. Перед применением алгоритма К-средних рекомендуется удалить или корректировать выбросы в данных.
- Работа с категориальными переменными: Если у вас есть категориальные переменные в данных, требуется их преобразование в числовую формат. Вы можете использовать методы, такие как One-Hot Encoding или Label Encoding.
- Обработка пропущенных значений: Если в ваших данных есть пропущенные значения, нужно решить, как с ними поступить. Вы можете удалить строки или столбцы с пропущенными значениями или заполнить их средними или медианными значениями в зависимости от контекста данных.
- Отбор признаков: При наличии большого количества признаков важно выбрать наиболее релевантные и информативные признаки для кластеризации. Вы можете использовать методы отбора признаков или снижения размерности данных, такие как анализ главных компонент (PCA) или методы выбора признаков на основе значимости.
- Обработка корреляций: Если у вас есть сильные корреляции между признаками, это может влиять на работу алгоритма К-средних. Рекомендуется выполнить анализ корреляций и решить, какие признаки оставить, исключить или объединить в один признак.
- Учет особенностей данных: Если в ваших данных есть особенности, такие как выборки с несбалансированным размером кластеров или наличие шума, следует учесть эти особенности при выборе количества кластеров и интерпретации результатов.

### 7.2.3 Процесс обучения

Основная идея алгоритма k-means заключается в том, чтобы минимизировать сумму квадратов ошибок (Sum of Squared Errors), то есть отклонения между объектами внутри кластеров и их центроидами. Сумма квадратов ошибок представлена следующим образом (Формула 7.6):

$$SSE = \sum_{i=1}^n \sum_{j=1}^k (x_{ij} - c_j)^2 \quad (7.6)$$

где  $n$  - общее количество точек данных,  $k$  - общее количество кластеров,  $x_{ij}$  -  $j$ -я компонента  $i$ -й точки данных, и  $c_j$  -  $j$ -й центроид кластера. В этом уравнении мы вычисляем квадрат расстояния между каждой точкой данных  $x_{ij}$  и соответствующим ей центроидом  $c_j$  для всех точек данных и кластеров. Затем мы суммируем эти квадраты расстояний, чтобы получить SSE.

Выбор оптимального количества кластеров в алгоритме K-Means может быть сложной задачей. Вот некоторые методы, которые можно использовать для определения оптимального числа кластеров:

- Метод локтя (elbow method): Этот метод заключается в вычислении суммы квадратов ошибок (SSE) для различных значений числа кластеров и выборе значения, при котором увеличение числа кластеров не приводит к существенному снижению SSE. График зависимости SSE от числа кластеров будет иметь форму локтя, и оптимальное число кластеров будет соответствовать точке, где SSE перестает значительно уменьшаться.
- Коэффициент силуэта (silhouette coefficient): Этот коэффициент используется для оценки качества кластеризации. Он учитывает среднее расстояние между объектами внутри кластера и среднее расстояние до ближайшего соседнего кластера. Значение коэффициента силуэта находится в диапазоне от -1 до 1, где более близкое к 1 значение указывает на лучшую кластеризацию. Оптимальное количество кластеров можно выбрать на основе максимального значения коэффициента силуэта.
- Метод gap statistic: Этот метод сравнивает значения SSE для фактических данных с ожидаемыми значениями SSE в случайной выборке. Чем больше разница между фактическими и ожидаемыми значениями SSE, тем лучше модель кластеризации. Оптимальное количество кластеров выбирается на основе максимального значения гап между фактическими и ожидаемыми значениями SSE.
- Экспертные знания: Иногда экспертные знания о предметной области могут помочь в выборе оптимального количества кластеров. Если вы знакомы с данными и ожидаете определенное количество групп или паттернов, вы можете использовать это знание для выбора числа кластеров.

Важно отметить, что выбор оптимального числа кластеров является относительным и может зависеть от конкретной задачи и данных. Рекомендуется использовать несколько методов оценки и сравнить результаты.

Алгоритм выполняет такие шаги до сходимости:

1. Инициализация: Выбираются случайные центроиды для каждого из кластеров. Центроид представляет собой точку в пространстве признаков, которая является центром кластера.
2. Присваивание кластера: Каждый объект из набора данных присваивается к ближайшему центроиду. Близость обычно измеряется с использованием евклидового расстояния, но можно использовать и другие метрики.
3. Обновление центроидов: Расчет нового центроида для каждого кластера путем вычисления среднего значения всех объектов, принадлежащих к данному кластеру.

Шаги 2-3 повторяются до тех пор, пока не будет достигнут критерий сходимости, например, пока изменение центроидов становится незначительным или количество итераций достигает заранее определенного предела.

#### 7.2.4 Оценка качества алгоритма

Оценка качества работы алгоритма К-средних (K-means) может быть выполнена с использованием нескольких метрик и методов. Вот некоторые распространенные способы оценки:

- Сумма квадратов ошибок (SSE): Это метрика, которая измеряет сумму квадратов расстояний между каждой точкой данных и центроидом ее кластера. Меньшее значение SSE указывает на лучшую кластеризацию. Однако SSE не является нормализованной метрикой и может зависеть от выбранного числа кластеров.

- Коэффициент силуэта (Silhouette coefficient): Это метрика, которая оценивает, насколько каждая точка данных хорошо соответствует своему собственному кластеру по сравнению с другими кластерами. Коэффициент силуэта принимает значения от -1 до 1, где ближе к 1 указывает на хорошую кластеризацию, а ближе к -1 - на неправильную кластеризацию.
- Индекс Дэвиса-Боулдина (Davies-Bouldin Index): Это индекс, который измеряет среднюю схожесть между кластерами и различие между кластерами. Меньшее значение индекса указывает на лучшую кластеризацию.
- Внутрикластерное расстояние и межкластерное расстояние: Можно также посмотреть на внутрикластерное расстояние (в среднем расстояние между точками внутри кластера) и межкластерное расстояние (среднее расстояние между центроидами кластеров). Хорошая кластеризация должна иметь маленькое внутрикластерное расстояние и большое межкластерное расстояние.
- Визуализация результатов: Визуализация кластеров может помочь визуально оценить качество работы алгоритма К-средних. Например, можно построить диаграмму рассеяния (англ. Scatter plot) и отобразить точки данных в пространстве признаков, окрашивая их в соответствии с принадлежностью кластерам. Пример визуального анализа с помощью диаграмм рассеяния приведен на рисунке 7.4.

### 7.2.5 Применение алгоритма

Алгоритм К-средних (K-means) имеет широкий спектр применений в различных областях. Некоторые из них включают:

- Кластерный анализ и сегментация данных: К-средних широко используется для кластеризации данных и сегментации на группы схожих объектов. Это может быть применено в маркетинге для сегментации клиентов на основе их предпочтений и поведения, в анализе данных для группировки схожих наблюдений и в биоинформатике для классификации геномных данных.
- Обработка изображений: К-средних может использоваться для сегментации изображений, разделения пикселей на различные группы в соответствии с их цветом или текстурой. Это может быть полезно, например, в распознавании образов, компьютерном зрении или сжатии изображений.
- Рекомендательные системы: В области рекомендательных систем К-средних может применяться для группировки пользователей или элементов в схожие кластеры. Это позволяет создавать персонализированные рекомендации, исходя из поведения или предпочтений пользователей.
- Анализ текстовых данных: К-средних может использоваться для анализа текстовых данных, например, для кластеризации документов по схожести содержания или группировки слов в тематические кластеры.
- Геоинформационные системы: В геоинформационных системах К-средних может быть применен для кластеризации пространственных данных, таких как точки на карте или географические регионы. Это может быть полезно для анализа распределения объектов, обнаружения паттернов или планирования маршрутов.
- Обнаружение аномалий: К-средних может использоваться для обнаружения аномальных наблюдений в данных. Аномальные точки могут быть удалены или выделены в

отдельные кластеры, что помогает в идентификации необычных событий или аномалий в различных областях, таких как финансы, кибербезопасность или медицинская диагностика.

Это только несколько примеров областей применения алгоритма K-средних, и он может быть использован во многих других сферах, где требуется кластеризация.

### 7.2.6 Плюсы и минусы алгоритма

#### Плюсы:

- Простота реализации и понимания: K-средних является простым и интуитивно понятным алгоритмом. Он основан на принципе минимизации суммы квадратов ошибок (SSE) и легко реализуется с помощью итеративного процесса.
- Высокая эффективность: K-средних имеет высокую вычислительную эффективность, особенно при больших объемах данных. Он может обрабатывать большие наборы данных относительно быстро и масштабируется для работы с большим количеством точек данных.
- Масштабируемость: Алгоритм K-средних хорошо масштабируется с увеличением количества кластеров или размера данных. Он может быть применен для различных задач кластеризации, начиная от небольшого числа кластеров до большого количества.
- Применимость к широкому спектру данных: K-средних может быть использован для различных типов данных, включая числовые данные, категориальные данные и даже текстовые данные. Он не зависит от распределения данных и может работать с разными типами признаков.
- Интерпретируемость результатов: Результаты K-средних могут быть легко интерпретированы и визуализированы. Кластеры, сформированные алгоритмом, представляют собой группы схожих объектов, что помогает в понимании структуры данных и выявлении паттернов.

#### Минусы:

- Чувствительность к выбору начальных центроидов: Результаты K-средних могут сильно зависеть от исходного выбора центроидов. Неправильная инициализация может привести к сходимости к локальному оптимуму, а не к глобальному оптимальному решению. Это может требовать множественного запуска алгоритма с разными начальными условиями.
- Зависимость от количества кластеров: Пользователь должен заранее определить количество кластеров K, что может быть сложной задачей. Неправильный выбор значения K может привести к неправильной кластеризации или объединению несхожих групп в один кластер.
- Предположение о выпуклости кластеров: Алгоритм K-средних предполагает, что кластеры являются выпуклыми. Выпуклость кластера является свойством структуры кластеризации данных. Она описывает, насколько кластер представляет собой выпуклую форму. Кластер считается выпуклым, если каждая точка внутри кластера находится внутри выпуклой оболочки кластера. Иначе говоря, в выпуклом кластере вы можете провести прямую линию из любой точки кластера в любую другую точку кластера, не покидая кластер. В случае, если данные имеют сложную форму или кластеры имеют различные формы и размеры, K-средних может давать неправильные результаты.

- Чувствительность к выбросам: К-средних чувствителен к выбросам в данных. Одиночные точки-выбросы могут сильно повлиять на положение центроидов и привести к искажению кластеризации.
- Нет гарантии глобального оптимума: Алгоритм К-средних может сойтись к локальному оптимуму, особенно при сложной структуре данных или плохой инициализации. В некоторых случаях может потребоваться использование более сложных алгоритмов кластеризации для достижения лучших результатов.

### 7.2.7 Реализация алгоритма в Python

K-means реализован в библиотеке sklearn. Для использования KMeans из библиотеки scikit-learn необходимо предварительно импортировать класс KMeans из модуля sklearn.cluster. Для задачи кластеризации с использованием K-means в библиотеке sklearn существует класс **KMeans**. Основные параметры модели:

- **n\_clusters**: количество кластеров, которое требуется найти. Это обязательный параметр, необходимо его указать при создании объекта KMeans.
- **init**: метод инициализации начальных центроидов. Может принимать значения «k-means++», «random» или массив начальных центроидов. По умолчанию используется метод «k-means++». K-means++ позволяет более равномерно распределить начальные центроиды по набору данных, что помогает избежать попадания в локальные минимумы и улучшает качество кластеризации.
- **n\_init**: количество запусков алгоритма с различными начальными центроидами. По умолчанию n\_init=10.
- **max\_iter**: максимальное количество итераций для сходимости алгоритма. По умолчанию max\_iter=300.
- **tol**: порог сходимости алгоритма. Если изменение SSE (суммы квадратов ошибок) между двумя последовательными итерациями меньше tol, алгоритм прекращает работу. По умолчанию tol=1e-4.

Класс KMeans также имеет методы fit(X) для обучения модели на данных X, predict(X) для присвоения меток кластеров новым данным X и transform(X) для преобразования данных X в матрицу расстояний до центроидов.

### 7.2.8 Вопросы для самопроверки

Какое предположение делает алгоритм K-Means о форме и размере кластеров?

1. Кластеры должны быть выпуклыми и иметь одинаковую дисперсию.
2. Кластеры могут иметь произвольную форму и размер.
3. Кластеры должны быть сферическими и иметь одинаковый радиус.
4. Кластеры должны быть линейно разделимыми.

Правильные ответы: 1

Какой параметр влияет на чувствительность алгоритма K-Means к выбросам?

1. Количество итераций (max\_iter).

2. Количество кластеров (`n_clusters`).
3. Метод инициализации начальных центроидов (`init`).
4. Порог сходимости алгоритма (`tol`).

Правильные ответы: 4

Как выбрать оптимальное количество кластеров в алгоритме K-Means?

1. Выбрать количество кластеров, равное количеству уникальных меток в данных.
2. Выбрать количество кластеров, равное среднему значению внутрикластерной дисперсии.
3. Использовать метод локтя (elbow method), оценивая изменение суммы квадратов ошибок при различном числе кластеров.
4. Определить количество кластеров на основе экспертных знаний о данных.

Правильные ответы: 3, 4

### 7.2.9 Резюме по разделу

K-means - это простой и популярный алгоритм кластеризации, который позволяет разделить набор данных на группы (кластеры) на основе их схожести. Краткое резюме по K-means:

- K-means итеративно присваивает точки к ближайшим центроидам и обновляет центроиды до сходимости.
- Алгоритм стремится минимизировать внутрикластерную сумму квадратов ошибок (SSE), которая измеряет сумму квадратов расстояний между точками и соответствующими центроидами.
- K-means требует заранее заданное количество кластеров (K), и выбор оптимального значения K является важной задачей при использовании K-means.
- Качество работы алгоритма может быть оценено с помощью метрик, таких как коэффициент силуэта или индекс Дэвиса-Боулдина.
- K-means может быть применен в различных областях, таких как анализ данных, сегментация клиентов, обработка изображений и многое другое. Однако K-means имеет свои ограничения, такие как чувствительность к начальной инициализации и форме кластеров, а также предположение о выпуклых кластерах и равной дисперсии.

## 7.3 DBSCAN

**Цель занятия:** ученик может применить алгоритм DBSCAN для решения задач кластеризации на подготовленных и неподготовленных данных.

**План занятия:**

- Визуальная демонстрация алгоритма
- Подготовка данных для алгоритма
- Процесс обучения
- Оценка качества алгоритма
- Применение алгоритма
- Плюсы и минусы алгоритма
- Реализация алгоритма в Python

### 7.3.1 Визуальная демонстрация алгоритма

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) - это алгоритм кластеризации данных, который определяет кластеры на основе плотности точек в пространстве данных. В отличие от других алгоритмов кластеризации, таких как k-средних или иерархическая кластеризация, DBSCAN может обнаруживать кластеры произвольной формы и способен обрабатывать шумовые точки.

Принцип работы DBSCAN заключается в следующем:

1. Определение окрестности точки: Для каждой точки в пространстве данных определяется окрестность, состоящая из всех точек, находящихся в заданном радиусе от данной точки.
2. Определение основной точки: Если в окрестности точки находится не менее  $\text{min\_samples}$  точек (включая саму точку), то эта точка считается основной точкой. На рисунке 7.7.1 основные точки обозначены красным цветом.
3. Формирование кластеров: Начиная с основных точек, алгоритм идентифицирует связанные основные точки и объединяет их в кластеры. Две точки считаются связанными, если есть путь от одной точки к другой через серию соседних точек, и каждая точка на этом пути также является основной.
4. Обработка граничных точек: Граничные точки находятся в окрестности основной точки, но сами не являются основными. Они могут принадлежать кластеру, если они связаны с основной точкой, но могут также оставаться неразмеченными. На рисунке 7.7.1 граничные точки обозначены синим цветом.
5. Идентификация шумовых точек: Точки, которые не являются ни основными, ни граничными, считаются шумовыми и не принадлежат ни к одному кластеру. На рисунке 7.7.1 шумовые точки обозначены темным цветом.

DBSCAN является одним из популярных алгоритмов кластеризации, особенно при работе с большими наборами данных и кластерами произвольной формы.

Визуальная демонстрация алгоритма представлена на рисунке 7.7.

На рисунке 7.7.1 красным выделены основные точки и их окрестности с параметром  $\text{min\_samples}=3$ . На рисунке 7.7.2 инициализируется выборка. На рисунке 7.7.3 алгоритм делит выборку на кластеры, отмечая основные точки кругами большего радиуса, граничные точки - кругами меньшего радиуса и шумовые точки - кругами темного цвета. На рисунке 7.7.4 показана возможность ведения алгоритмом DBSCAN кластеров сложной формы.

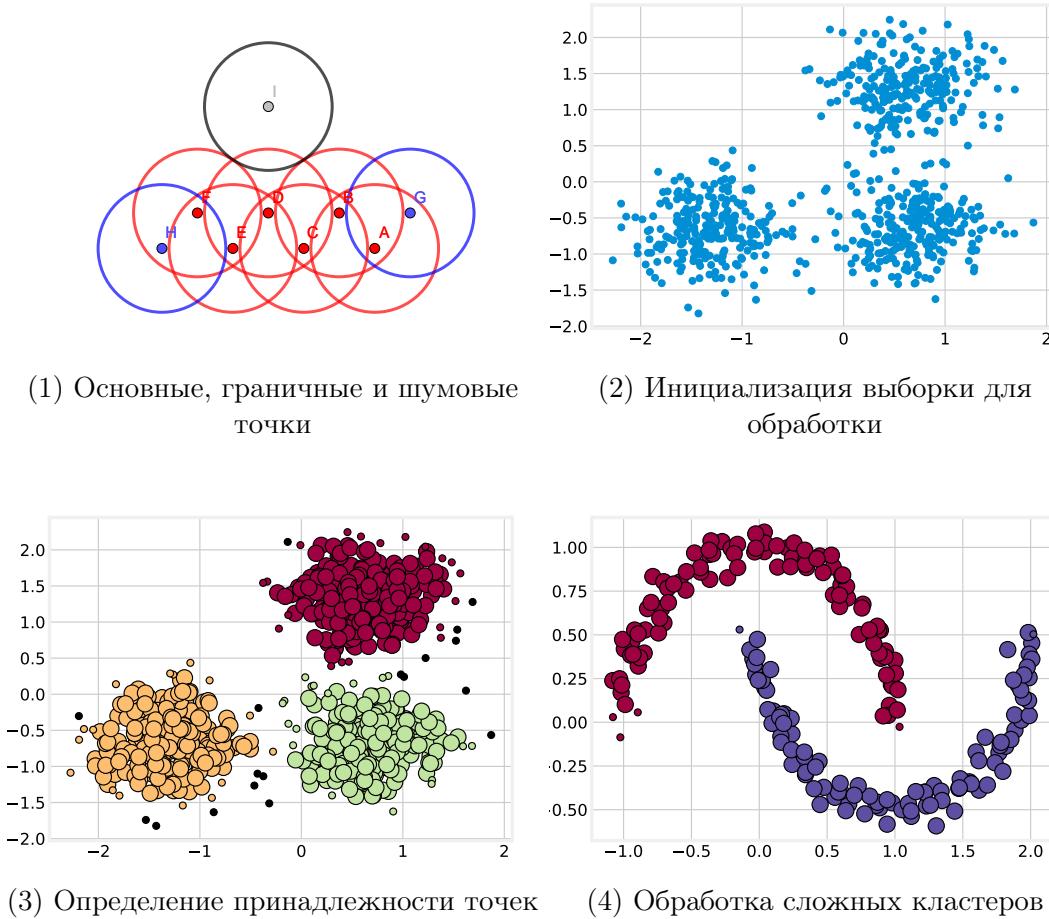


Рисунок 7.7: Демонстрация принципа работы алгоритма DBSCAN

### 7.3.2 Подготовка данных для алгоритма

Общий процесс подготовки данных для использования с алгоритмом DBSCAN:

- **Масштабирование данных:** Рекомендуется масштабировать признаки входных данных перед применением DBSCAN, особенно если признаки имеют различные шкалы или единицы измерения. Обычно используется стандартизация или нормализация данных для приведения их к общей шкале.
- **Удаление выбросов:** Выбросы могут оказывать влияние на результаты DBSCAN, поскольку алгоритм опирается на плотность точек. Перед применением DBSCAN рекомендуется удалить или корректировать выбросы в данных.
- **Работа с категориальными переменными:** Если у вас есть категориальные переменные в данных, требуется их преобразование в числовую формат. Вы можете использовать методы, такие как One-Hot Encoding или Label Encoding, чтобы закодировать категориальные переменные перед применением DBSCAN.
- **Обработка пропущенных значений:** Если в ваших данных есть пропущенные значения, нужно решить, как с ними поступить. Вы можете удалить строки или столбцы с пропущенными значениями или заполнить их средними или медианными значениями в зависимости от контекста данных.

- Отбор признаков: При наличии большого количества признаков важно выбрать наиболее релевантные и информативные признаки для кластеризации с помощью DBSCAN. Вы можете использовать методы отбора признаков или снижения размерности данных, такие как анализ главных компонент (PCA) или методы выбора признаков на основе значимости.
- Обработка корреляций: Если у вас есть сильные корреляции между признаками, это может влиять на работу DBSCAN. Рекомендуется выполнить анализ корреляций и решить, какие признаки оставить, исключить или объединить в один признак.
- Учет особенностей данных: Если в ваших данных есть особенности, такие как выборки с несбалансированным размером кластеров или наличие шума, следует учесть эти особенности при выборе параметров DBSCAN и интерпретации результатов.

### 7.3.3 Процесс обучения

Процесс обучения алгоритма DBSCAN состоит из следующих шагов:

1. Задание параметров: Прежде чем приступить к обучению DBSCAN, необходимо задать параметры алгоритма. Главными параметрами являются:
  - eps (epsilon): радиус окрестности, в пределах которой точки считаются соседними.
  - min\_samples: минимальное количество точек в окрестности, необходимое для определения основной точки.Значения этих параметров должны быть выбраны в соответствии с характеристиками данных и требуемым уровнем плотности точек.
2. Вычисление плотности: Алгоритм DBSCAN анализирует плотность точек в пространстве данных. Для каждой точки вычисляется число соседей, находящихся в пределах заданного радиуса eps.
3. Определение типов точек: В зависимости от числа соседей каждая точка может быть классифицирована как:
  - Основная точка (core point): точка, для которой число соседей в окрестности eps превышает или равно min\_samples.
  - Границчная точка (border point): точка, для которой число соседей в окрестности eps меньше min\_samples, но находится в окрестности основной точки.
  - Шумовая точка (noise point): точка, для которой число соседей в окрестности eps меньше min\_samples и не находится в окрестности основной точки.
4. Формирование кластеров: DBSCAN идентифицирует связанные основные точки и объединяет их в кластеры. Две точки считаются связанными, если есть путь от одной точки к другой через серию соседних точек, и каждая точка на этом пути также является основной. В результате формируются кластеры различной формы и размеров.
5. Маркировка граничных точек: Граничные точки, которые не принадлежат ни одному кластеру, могут быть отмечены соответствующим образом или оставаться без метки.

Важно отметить, что DBSCAN не требует явного обучения или итераций, как в случае с алгоритмами, основанными на центроидах, например, K-средних. Он основывается на анализе плотности точек и обнаружении связанных областей в данных. Поэтому процесс работы DBSCAN не включает фазы обучения и тестирования, а только параметризацию и применение алгоритма к данным.

### 7.3.4 Оценка качества алгоритма

Оценка качества работы алгоритма DBSCAN может быть выполнена с использованием следующих метрик и методов:

- Коэффициент силуэта (Silhouette coefficient): Как и в случае с алгоритмом K-средних, коэффициент силуэта может использоваться для оценки качества кластеризации с помощью DBSCAN. Он измеряет, насколько каждая точка данных хорошо соответствует своему кластеру по сравнению с другими кластерами. Значения коэффициента силуэта варьируются от -1 до 1, где ближе к 1 указывает на хорошую кластеризацию, ближе к -1 - на неправильную кластеризацию, а близость к 0 означает, что точки могут находиться на границе между кластерами.
- Количество обнаруженных кластеров: DBSCAN не требует задания числа кластеров заранее. Однако количество обнаруженных кластеров может быть использовано в качестве метрики для оценки работы алгоритма. Слишком большое или слишком маленькое количество кластеров может быть признаком неправильной кластеризации или недостаточной гибкости алгоритма.
- Визуализация результатов: Визуализация кластеров, полученных с помощью DBSCAN, может быть полезным способом оценки их качества. Аналогично K-средним, можно построить диаграмму рассеяния и окрасить точки данных в соответствии с принадлежностью кластерам. Визуальное исследование может помочь определить, насколько хорошо алгоритм обнаруживает плотные области данных и как обрабатывает шумовые точки.
- Отношение шума к кластерам: DBSCAN также может быть оценен по отношению шумовых точек к обнаруженным кластерам. Слишком большое количество шумовых точек может быть признаком плохой кластеризации или неправильно выбранных параметров алгоритма.

Важно отметить, что DBSCAN, в отличие от K-средних, может обнаруживать выбросы как шумовые точки, и его оценка может отличаться в зависимости от природы данных и параметров алгоритма.

### 7.3.5 Применение алгоритма

Алгоритм DBSCAN имеет широкий спектр применений в различных областях. Некоторые из них включают:

- Кластерный анализ и сегментация данных: DBSCAN используется для кластеризации данных и сегментации на группы схожих объектов. Он позволяет обнаруживать плотные области в данных и отделять их от разреженных областей. Применяется, например, для сегментации географических данных, обнаружения сообществ в социальных сетях или анализа покупательского поведения.
- Обнаружение выбросов и аномалий: DBSCAN может использоваться для обнаружения выбросов и аномалий в данных. Шумовые точки, которые не принадлежат ни к одному кластеру, могут быть идентифицированы как выбросы или аномалии. Применяется, например, для обнаружения аномальных транзакций в финансовых данных или необычных медицинских записей.
- Геоинформационные системы: В геоинформационных системах DBSCAN может быть использован для кластеризации пространственных данных, таких как точки на карте,

географические регионы или географические маршруты. Применяется, например, для анализа распределения объектов на карте, выявления географических паттернов или планирования оптимальных маршрутов.

- Сегментация изображений: DBSCAN может быть применен для сегментации изображений, разделения пикселей на группы в соответствии с их пространственным расположением. Применяется, например, в компьютерном зрении для выделения объектов на изображении или сегментации текстурных областей.
- Рекомендательные системы: DBSCAN может использоваться в рекомендательных системах для группировки пользователей или элементов на основе их пространственной близости или схожести. Это позволяет создавать персонализированные рекомендации или группировать похожие элементы в рекомендациях.
- Анализ сетей и связей: DBSCAN может быть применен для анализа структуры сетей или связей между объектами, например, он позволяет выявлять сообщества в социальных сетях.

Это только несколько примеров областей применения алгоритма DBSCAN, и он может быть использован во многих других сферах, где требуется кластеризация.

### 7.3.6 Плюсы и минусы алгоритма

#### Плюсы:

- Способность обнаруживать кластеры произвольной формы: DBSCAN не требует заранее заданного числа кластеров и способен обнаруживать кластеры произвольной формы. Он основывается на плотности точек данных и может определить плотные области в данных, независимо от их формы.
- Работа с шумом и выбросами: DBSCAN может эффективно обрабатывать шумовые точки и выбросы в данных. Он идентифицирует точки, которые не принадлежат ни к одному кластеру, и относит их к категории шума. Это делает DBSCAN устойчивым к выбросам и способным обрабатывать данные с неопределенными или неструктурированными областями.
- Не требует предварительной спецификации числа кластеров: DBSCAN не требует заранее заданного числа кластеров, в отличие от некоторых других алгоритмов кластеризации. Это позволяет ему автоматически определять число кластеров на основе плотностной структуры данных.
- Гибкость в определении плотности: DBSCAN позволяет гибко настраивать параметры, определяющие плотность кластеров. Можно задать радиус окрестности точки (*epsilon*) и минимальное количество точек в окрестности (*min\_samples*), чтобы определить, что точка является ядром кластера. Это позволяет адаптировать DBSCAN под различные плотностные характеристики данных.
- Эффективность для больших объемов данных: DBSCAN может быть эффективным для обработки больших объемов данных. Он использует индексацию и структуры данных, такие как деревья, для ускорения поиска ближайших соседей и определения плотных областей.
- Способность к обнаружению выбросов и аномалий: Благодаря способности DBSCAN идентифицировать шумовые точки, он может быть использован для обнаружения выбросов и аномалий в данных. Это полезно во многих областях, таких как обнаружение мошенничества, анализ аномалий или медицинская диагностика.

### Минусы:

- Чувствительность к выбору параметров: DBSCAN требует настройки двух основных параметров - радиуса окрестности (`epsilon`) и минимального количества точек в окрестности (`min_samples`). Выбор подходящих значений этих параметров может быть нетривиальной задачей и требует предварительного анализа данных или итеративного подбора. Неправильный выбор параметров может привести к неправильной классификации или объединению кластеров.
- Сложность работы с данными разной плотности: DBSCAN может иметь проблемы с кластеризацией данных, содержащих кластеры разной плотности. В случае, когда кластеры имеют значительно различающуюся плотность, может возникнуть сложность в правильном определении радиуса окрестности (`epsilon`) и минимального количества точек в окрестности (`min_samples`) для каждого кластера.
- Зависимость от плотности данных: DBSCAN может столкнуться с проблемой в случае данных с неравномерной плотностью. В областях, где плотность данных существенно меняется, алгоритм может иметь трудности с корректным определением границ кластеров.
- Сложности при работе с высокоразмерными данными: DBSCAN имеет сложности при работе с высокоразмерными данными, так как пространство высокой размерности может быть разреженным и иметь проблемы с определением плотных областей. Это может привести к неправильной кластеризации или потере информации.
- Отсутствие поддержки иерархической кластеризации: DBSCAN не поддерживает иерархическую кластеризацию, то есть невозможно получить иерархию кластеров с различными уровнями детализации.

### 7.3.7 Реализация алгоритма в Python

Алгоритм DBSCAN реализован в библиотеке scikit-learn и доступен через класс `DBSCAN` из модуля `sklearn.cluster`. Для использования `DBSCAN` в `scikit-learn` необходимо предварительно импортировать этот класс.

Вот основные параметры модели `DBSCAN`:

- **`eps`**: радиус окрестности, в пределах которой должно находиться минимальное количество точек, чтобы рассматриваемая точка была считана основной (core point). Это обязательный параметр, который следует выбирать с учетом особенностей данных.
- **`min_samples`**: минимальное количество точек, которые должны находиться в радиусе окрестности (`eps`), чтобы рассматриваемая точка была считана основной (core point). По умолчанию `min_samples=5`.
- **`metric`**: используемая метрика для измерения расстояния между точками. Может принимать различные значения, такие как «`euclidean`» (евклидово расстояние), «`manhattan`» (манхэттенское расстояние) и другие. По умолчанию `metric=«euclidean»`.
- **`algorithm`**: используемый алгоритм для выполнения `DBSCAN`. Может принимать значения «`auto`», «`ball_tree`», «`kd_tree`» или «`brute`». По умолчанию `algorithm=«auto»`, что позволяет автоматически выбрать наиболее эффективный алгоритм на основе входных данных.
- **`leaf_size`**: размер листа, используемый в алгоритмах «`ball_tree`» или «`kd_tree`». Этот параметр влияет на скорость построения дерева и использование памяти. По умолчанию `leaf_size=30`.

- **metric\_params:** дополнительные параметры, которые могут быть переданы используемой метрике. Например, для метрики Минковского (Minkowski) можно указать значение параметра  $p$  через `metric_params='p': 2`.

Класс DBSCAN также имеет методы `fit(X)` для обучения модели на данных  $X$ , `fit_predict(X)` для кластеризации данных и присвоения меток кластеров, а также методы `fit_transform(X)` и `transform(X)` для преобразования данных  $X$  в матрицу расстояний до основных точек и кластеризации соответственно.

### 7.3.8 Вопросы для самопроверки

Какие типы точек присутствуют в алгоритме DBSCAN (один ответ)?

1. Основные (core points), граничные (border points) и шумовые (outliers)
2. Центроиды (centroids), границы (boundaries) и фоновые (background)
3. Критические (critical points), периферийные (peripheral points) и шумовые (outliers)
4. Произвольные (arbitrary points), отдельные (isolated points) и фоновые (background)

Правильные ответы: 1

Какие основные параметры нужно задать для алгоритма DBSCAN (один ответ)?

1. Количество кластеров и метод инициализации центроидов
2. Радиус окрестности и минимальное количество точек в окрестности
3. Максимальное количество итераций и порог сходимости
4. Метрика расстояния и размер листа для алгоритма кластеризации

Правильные ответы: 2

Плюсы алгоритма DBSCAN (четыре ответа)?

1. Гибкость в определении плотности
2. Зависимость от плотности данных
3. Отсутствие поддержки иерархической кластеризации
4. Работа с шумом и выбросами
5. Сложности при работе с высокоразмерными данными
6. Сложность работы с данными разной плотности
7. Способность к обнаружению выбросов и аномалий
8. Способность обнаруживать кластеры произвольной формы

Правильные ответы: 1, 4, 7, 8

### 7.3.9 Резюме по разделу

DBSCAN (Density-Based Spatial Clustering of Applications with Noise) - это алгоритм кластеризации, который основывается на плотности данных. Вместо предварительно заданного числа кластеров, DBSCAN автоматически определяет количество и форму кластеров, исходя из плотности точек данных.

Основные преимущества DBSCAN:

- Способность обнаруживать кластеры произвольной формы: DBSCAN может успешно обрабатывать кластеры произвольной формы, включая кластеры с несферическими или неравными размерами.
- Не нужно указывать количество кластеров: DBSCAN автоматически определяет количество кластеров на основе плотности данных, что делает его особенно полезным при отсутствии заранее известной информации о количестве кластеров.
- Устойчивость к выбросам: DBSCAN способен обнаруживать и отделять выбросы (точки, не принадлежащие к какому-либо кластеру) от кластеров, что помогает в обработке шумных данных.

Некоторые ограничения DBSCAN:

- Зависимость от параметров: DBSCAN требует задания двух параметров - радиуса окрестности (eps) и минимального числа точек в окрестности (min\_samples). Неправильные выбранные значения этих параметров могут привести к нежелательным результатам.
- Чувствительность к плотности данных: DBSCAN может столкнуться с трудностями, когда в данных присутствует значительная разница в плотности между кластерами. В таких случаях, определение подходящих значений параметров может быть сложным.
- Высокая вычислительная сложность: Алгоритм DBSCAN может быть вычислительно требовательным при работе с большими объемами данных, особенно если данные не были предварительно отфильтрованы или преобразованы.

В целом, DBSCAN является мощным инструментом для кластеризации данных, особенно в случаях, когда форма и количество кластеров неизвестны заранее, и когда важно выделение выбросов и обработка шумных данных. Однако правильный выбор параметров и управление вычислительной сложностью могут потребовать дополнительных усилий и экспериментов.

## 7.4 Агломеративная кластеризация

**Цель занятия:** ученик может применить алгоритм агломеративной кластеризации для решения задач кластеризации на подготовленных и неподготовленных данных.

**План занятия:**

- Визуальная демонстрация алгоритма
- Подготовка данных для алгоритма
- Процесс обучения
- Оценка качества алгоритма
- Применение алгоритма
- Плюсы и минусы алгоритма
- Реализация алгоритма в Python

### 7.4.1 Визуальная демонстрация алгоритма

Агломеративная кластеризация — это один из методов кластерного анализа, который используется для группировки объектов в иерархическую структуру. Он начинается с того, что каждый объект представляется отдельным кластером, а затем последовательно объединяет наиболее похожие кластеры до тех пор, пока не будет получен единственный кластер, содержащий все объекты. Визуально этот процесс можно представить следующим образом (Рисунок 7.8). На каждом шаге мы сливаем два наиболее близких кластера, а также добавляем новый уровень в дендрограмму.

Дендрограмма - это графическое представление результатов иерархической кластеризации. Она представляет собой дерево-структуру, где каждый узел представляет кластер или группу объектов, а ветви указывают на близость или расстояние между кластерами.

На дендрограмме ось Y представляет меру расстояния или сходства между кластерами. Чем выше на оси Y находится точка слияния кластеров, тем дальше они друг от друга или менее похожи. Можно провести горизонтальную линию через дендрограмму, чтобы определить, какие кластеры объединяются на каждом уровне.

Ось X дендрограммы представляет собой список объектов или кластеров. Каждая точка на оси X соответствует отдельному объекту или кластеру. Путем анализа дендрограммы можно определить, какие объекты или кластеры объединяются на каждом уровне иерархии.

Дендрограммы широко используются в кластерном анализе для визуализации и интерпретации результатов. Они помогают исследователям определить оптимальное количество кластеров или иерархическую структуру данных. Также дендрограммы могут быть полезны при принятии решений о разделении кластеров на разных уровнях детализации, а также для анализа подобия или сходства между кластерами.

После того, как построен единственный кластер и дендрограмма, по дендрограмме можно выбрать количество кластеров, выбрав соответствующий уровень дендрограммы для разбиения на кластеры (на рисунке 7.8.8 один из вариантов разбиения выделен красным пунктиром).

### 7.4.2 Подготовка данных для алгоритма

Ниже представлен общий процесс подготовки данных для использования с алгоритмом агломеративной кластеризации:

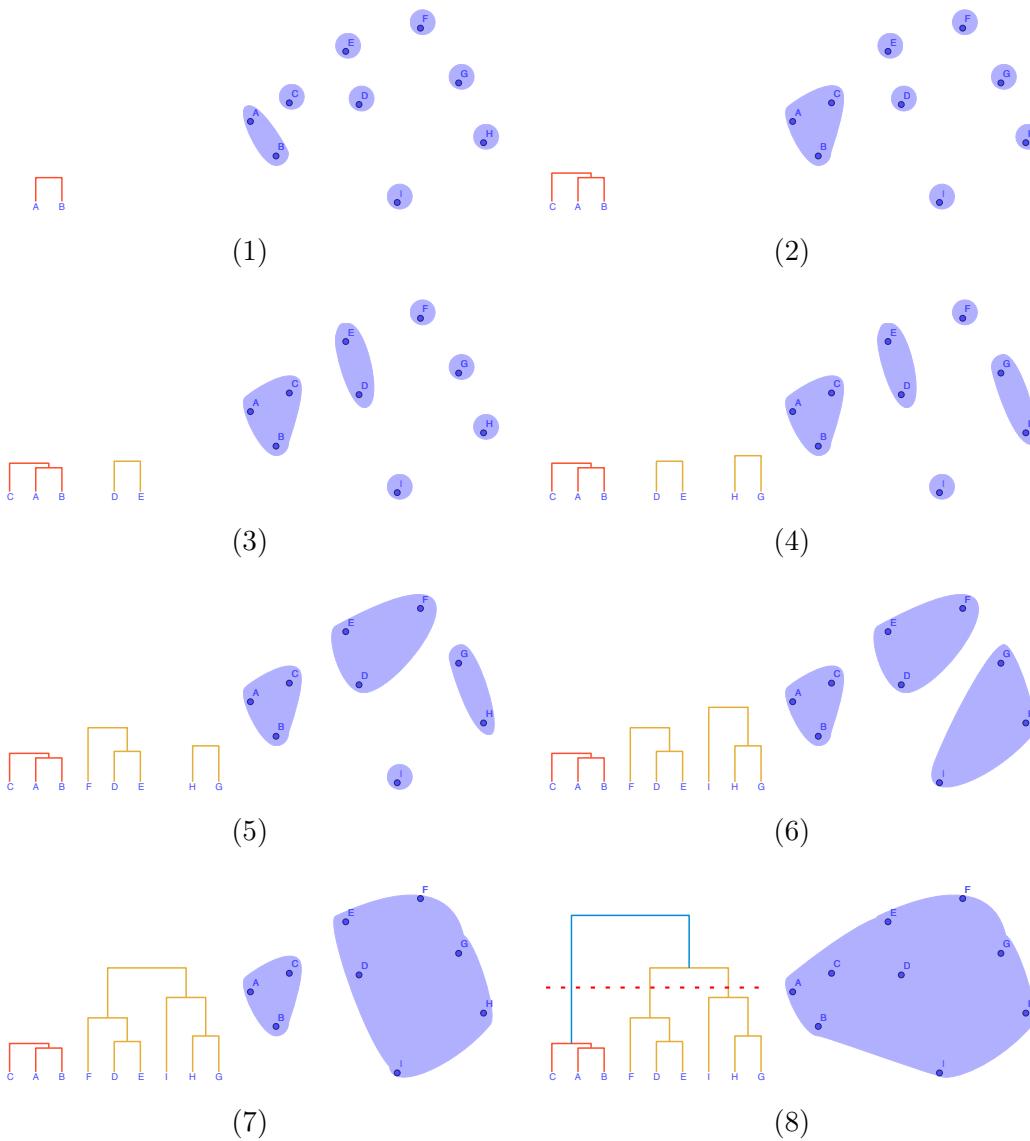


Рисунок 7.8: Пошаговый процесс построения дендрограммы и единственного кластера для агломеративной кластеризации

- **Масштабирование данных:** Рекомендуется масштабировать признаки входных данных перед применением агломеративной кластеризации, особенно если признаки имеют различные шкалы или единицы измерения. Обычно используется стандартизация или нормализация данных для приведения их к общей шкале.
- **Обработка категориальных переменных:** Если у вас есть категориальные переменные в данных, требуется их преобразование в числовой формат. Вы можете использовать методы, такие как One-Hot Encoding или Label Encoding, чтобы закодировать категориальные переменные перед применением агломеративной кластеризации.
- **Обработка пропущенных значений:** Если в ваших данных есть пропущенные значения, нужно решить, как с ними поступить. Вы можете удалить строки или столбцы с пропущенными значениями или заполнить их средними или медианными значениями в зависимости от контекста данных.

- Отбор признаков: При наличии большого количества признаков важно выбрать наиболее релевантные и информативные признаки для кластеризации с помощью агломеративной кластеризации. Вы можете использовать методы отбора признаков или снижения размерности данных, такие как анализ главных компонент (PCA) или методы выбора признаков на основе значимости.
- Обработка корреляций: Если у вас есть сильные корреляции между признаками, это может влиять на работу агломеративной кластеризации. Рекомендуется выполнить анализ корреляций и решить, какие признаки оставить, исключить или объединить в один признак.
- Учет особенностей данных: Если в ваших данных есть особенности, такие как выборки с несбалансированным размером кластеров или наличие шума, следует учесть эти особенности при выборе параметров агломеративной кластеризации и интерпретации результатов.

### 7.4.3 Процесс обучения

Вот основные шаги агломеративной кластеризации:

1. Инициализация: Каждый объект представляется в отдельном кластере.
2. Вычисление матрицы расстояний: Вычисляется матрица расстояний между всеми парами кластеров. Расстояние может быть определено различными способами, например, евклидово расстояние или корреляция.
3. Объединение кластеров: Наиболее похожие кластеры объединяются в один кластер на основе определенного критерия объединения. Распространенными критериями являются минимальное расстояние (single linkage), максимальное расстояние (complete linkage) или среднее расстояние (average linkage) между кластерами.
4. Обновление матрицы расстояний: Матрица расстояний обновляется, чтобы отразить новые расстояния между объединенными кластерами.
5. Повторение шагов 3 и 4: Шаги объединения кластеров и обновления матрицы расстояний повторяются до тех пор, пока все объекты не будут объединены в один кластер.
6. Формирование дендрограммы: В результате агломеративной кластеризации можно получить дендрограмму, которая представляет собой дерево объединений кластеров. По оси X дендрограммы отображаются объекты или кластеры, а по оси Y отображается мера расстояния или сходства.

Агломеративная кластеризация не требует заранее заданного числа кластеров и позволяет исследовать структуру данных на разных уровнях детализации. Этот метод широко используется в различных областях, включая анализ данных, биоинформатику и т. д.

### 7.4.4 Оценка качества алгоритма

Оценка качества алгоритма агломеративной кластеризации может быть выполнена с использованием различных метрик и методов. Вот некоторые распространенные меры, которые могут быть использованы для оценки качества агломеративной кластеризации:

- Внутрикластерное расстояние: Эта метрика измеряет среднее расстояние между объектами внутри каждого кластера. Чем меньше значение внутrikластерного расстояния, тем лучше.

- Межкластерное расстояние: Эта метрика измеряет расстояние между центроидами или центрами масс различных кластеров. Чем больше значение межкластерного расстояния, тем лучше.
- Коэффициент силуэта: Коэффициент силуэта оценивает качество кластеризации, учитывая как близость объектов внутри своего кластера, так и удаленность от соседних кластеров. Значение коэффициента силуэта находится в диапазоне от -1 до 1, где значения ближе к 1 указывают на хорошую кластеризацию.

Могут использоваться и другие метрики в соответствии с контекстом оценки качества алгоритма агломеративной кластеризации.

#### 7.4.5 Применение алгоритма

Алгоритм агломеративной кластеризации широко применяется в различных областях анализа данных и машинного обучения. Вот некоторые области, где алгоритм агломеративной кластеризации может быть полезным:

- Маркетинг и сегментация потребителей: Агломеративная кластеризация может использоваться для идентификации групп потребителей с похожими предпочтениями, поведением или демографическими характеристиками. Это позволяет компаниям создавать более целевые маркетинговые стратегии и персонализированные предложения.
- Биология и генетика: Агломеративная кластеризация может быть применена для анализа генетических данных и выявления генетических паттернов или групп, связанных с определенными заболеваниями или фенотипами. Это помогает в понимании генетической структуры и классификации образцов.
- Обработка естественного языка: В анализе текстовых данных агломеративная кластеризация может использоваться для группировки документов или текстовых фрагментов по сходству содержания. Это может быть полезно, например, для создания тематических кластеров или выявления паттернов в текстовых данных.
- Обработка изображений и компьютерное зрение: Агломеративная кластеризация может применяться для сегментации изображений по сходству пикселей или объектов на изображении. Это используется, например, для распознавания образов или выделения регионов интереса на изображении.
- Социальные сети и анализ социальных данных: Агломеративная кластеризация может помочь в анализе социальных сетей и идентификации групп пользователей с похожими интересами, связями или поведением. Это полезно для рекомендательных систем, персонализации контента и анализа влияния в социальных сетях.

Это только несколько примеров областей применения агломеративной кластеризации. Алгоритм может быть использован во многих других областях, где требуется группировка и классификация данных на основе их сходства.

#### 7.4.6 Плюсы и минусы алгоритма

##### Плюсы

- Простота реализации: Агломеративная кластеризация является относительно простым алгоритмом, который легко реализовать и понять. Он основывается на простых принципах объединения ближайших точек или кластеров на каждом шаге.

- Гибкость и масштабируемость: Алгоритм агломеративной кластеризации может быть применен к различным типам данных и может обрабатывать большие объемы данных. Он может работать с различными мерами расстояния и сходства, что позволяет адаптировать его к различным задачам.
- Иерархическая структура: Агломеративная кластеризация создает иерархическую структуру кластеров, которая позволяет анализировать данные на разных уровнях детализации. Это позволяет получить информацию о более общих группах и более специфичных подгруппах данных.
- Способность обнаруживать кластеры различных форм и размеров: Алгоритм агломеративной кластеризации способен обнаруживать кластеры различных форм, размеров и плотности. Он может обрабатывать кластеры с произвольной формой и даже с неоднородной плотностью.
- Возможность визуализации: Иерархическая структура, созданная алгоритмом агломеративной кластеризации, может быть визуализирована в виде дендрограммы. Дендрограмма представляет собой графическое представление иерархии кластеров, что упрощает интерпретацию результатов.

### Минусы

- Высокая вычислительная сложность: Агломеративная кластеризация имеет квадратичную сложность по времени, что делает ее менее эффективной для больших наборов данных. Вычисление расстояний между всеми парами точек может быть ресурсоемкой операцией.
- Чувствительность к выбору меры расстояния: Выбор подходящей меры расстояния является критическим для успешной работы агломеративной кластеризации. Разные меры могут давать различные результаты, и некорректный выбор меры может привести к искажению кластерной структуры.
- Отсутствие контроля над размером кластеров: Алгоритм агломеративной кластеризации не предоставляет явного способа контроля над размером получаемых кластеров. Размеры кластеров могут существенно отличаться, и в некоторых случаях это может быть проблемой.

#### 7.4.7 Реализация алгоритма в Python

Агломеративная кластеризация реализована в библиотеке scikit-learn и доступна через класс AgglomerativeClustering из модуля sklearn.cluster.

Основные параметры модели AgglomerativeClustering:

- **n\_clusters:** количество кластеров, которые требуется сформировать. Это обязательный параметр.
- **affinity:** метрика, используемая для вычисления расстояния между точками. Может принимать различные значения, такие как «euclidean» (евклидово расстояние), «manhattan» (манхэттенское расстояние), «cosine» (косинусное расстояние) и другие.
- **linkage:** метод объединения кластеров. Может принимать значения «ward», «complete», «average» и «single».
- **distance\_threshold:** пороговое расстояние, при котором прекращается объединение кластеров. Если не указано, то все кластеры будут объединены.

Класс AgglomerativeClustering также имеет методы fit(X) для обучения модели на данных X и fit\_predict(X) для кластеризации данных и присвоения меток кластеров.

### 7.4.8 Вопросы для самопроверки

Какова основная идея агломеративной кластеризации (один ответ)?

1. Разделение данных на заранее заданное число кластеров.
2. Объединение ближайших элементов в иерархическую структуру.
3. Поиск наиболее отдаленных элементов для формирования кластеров.
4. Применение деревьев решений для расчета расстояния между элементами.

Правильные ответы: 2

Какая метрика используется для вычисления расстояния между точками в агломеративной кластеризации?

1. Евклидово расстояние
2. Манхэттенское расстояние
3. Косинусное расстояние
4. Все вышеперечисленные

Правильные ответы: 4

Выберите плюсы агломеративной кластеризации (четыре ответа):

- Возможность визуализации
- Высокая вычислительная сложность
- Гибкость и масштабируемость
- Простота реализации
- Способность обнаруживать кластеры различных форм и размеров
- Трудность интерпретации результатов

Правильные ответы: 1, 3, 4, 5

### 7.4.9 Резюме по разделу

Агломеративная кластеризация - это метод машинного обучения, который используется для группировки объектов на основе сходства между ними. Этот метод начинает с каждого объекта, рассматриваемого как отдельный кластер, и объединяет их по мере того, как они становятся более похожими друг на друга. Для этого используются различные меры сходства, такие как евклидово расстояние или корреляция. Результатом агломеративной кластеризации является дерево, называемое дендрограммой, которое показывает, как объекты были объединены в кластеры.

## 7.5 Метод главных компонент

**Цель занятия:** ученик может применить метод главных компонент (PCA) для решения задач снижения размерности на подготовленных и неподготовленных данных.

**План занятия:**

- Визуальная демонстрация алгоритма
- Подготовка данных для алгоритма
- Процесс обучения
- Оценка качества алгоритма
- Применение алгоритма
- Плюсы и минусы алгоритма
- Реализация алгоритма в Python

Метод главных компонент (англ. Principal Component Analysis, PCA) — это статистический метод, используемый для уменьшения размерности данных и извлечения наиболее значимых информационных характеристик из множества переменных. Он позволяет сжать исходный набор данных, сохраняя при этом наибольшее количество информации. Он основан на линейном преобразовании данных с целью найти новые оси (главные компоненты), вдоль которых данные имеют наибольшую изменчивость. Главные компоненты являются линейными комбинациями исходных признаков и ортогональны (перпендикулярны) друг другу. Они упорядочены по убыванию объясненной ими дисперсии данных.

Для объяснения принципов работы PCA необходимо рассмотреть некоторые понятия.

Дисперсия - это мера разброса или изменчивости случайной величины. Она показывает, насколько сильно значения случайной величины отклоняются от ее среднего значения.

Для случайной величины  $X$  с функцией вероятности (или функцией плотности вероятности)  $f(x)$  и средним значением  $\mu$ , дисперсия  $\sigma^2$  определяется следующим образом:

$$\sigma^2 = E[(X - \mu)^2] = \int (x - \mu)^2 f(x) dx$$

где  $E[\cdot]$  обозначает математическое ожидание, а интеграл берется по всем возможным значениям  $x$  случайной величины.

Дисперсия является положительным числом и представляет собой среднюю квадратическую разницу между значениями случайной величины и ее средним значением. Чем больше дисперсия, тем больше вариативность или разброс значений случайной величины.

В контексте PCA, объясненная дисперсия относится к количеству дисперсии в исходных данных, которая объясняется каждой главной компонентой.

Вычисление Среднее<sub>PC1</sub> (среднего значения первой главной компоненты) осуществляется путем суммирования всех значений первой главной компоненты и деления этой суммы на общее количество наблюдений.

Математически, Среднее<sub>PC1</sub> можно вычислить следующим образом:

$$\text{Среднее}_{\text{PC1}} = \frac{\sum_{i=1}^n \text{PC1}_i}{n}$$

где  $\text{PC1}_i$  - значения первой главной компоненты для каждого наблюдения, а  $n$  - общее количество наблюдений.

Объясненная дисперсия первой главной компоненты может быть записана как:

$$\text{Дисперсия}_{\text{PC}1} = \frac{\sum_{i=1}^n (\text{PC}1_i - \text{Среднее}_{\text{PC}1})^2}{n}$$

где  $\text{PC}1_i$  - значения первой главной компоненты для каждого наблюдения,  $\text{Среднее}_{\text{PC}1}$  - среднее значение первой главной компоненты, и  $n$  - общее количество наблюдений.

Аналогично может быть вычислена объясненная дисперсия для других главных компонент.

Объясненная дисперсия позволяет определить, какую долю дисперсии данных объясняет каждая главная компонента. Чем больше объясненная дисперсия, тем больше информации оригинальных данных сохраняется в соответствующей главной компоненте.

PCA находит главные компоненты путем вычисления собственных значений и собственных векторов ковариационной матрицы данных.

Ковариация — это мера статистической зависимости между двумя случайными переменными. Она измеряет, насколько две переменные изменяются вместе. Ковариация может быть положительной, если две переменные сильно положительно коррелируют (увеличение одной переменной связано с увеличением другой), отрицательной, если они сильно отрицательно коррелируют (увеличение одной переменной связано с уменьшением другой), или близкой к нулю, если между ними нет линейной зависимости (Рисунок 7.9).

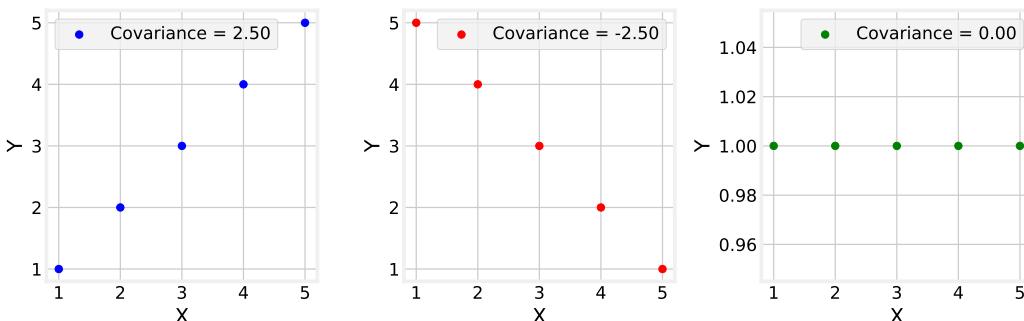


Рисунок 7.9: Положительная, отрицательная и нулевая ковариация переменных  $X$  и  $Y$

Ковариация двух случайных переменных  $X$  и  $Y$  может быть записана следующим образом:

$$\text{cov}(X, Y) = E[(X - E[X])(Y - E[Y])]$$

где  $E[X]$  и  $E[Y]$  обозначают математические ожидания (средние значения) переменных  $X$  и  $Y$  соответственно.

Ковариационная матрица - это квадратная матрица, которая содержит ковариации между парами случайных переменных. Для  $n$  случайных переменных  $X_1, X_2, \dots, X_n$ , ковариационная матрица  $C$  определяется следующим образом:

$$C = \begin{bmatrix} \text{cov}(X_1, X_1) & \text{cov}(X_1, X_2) & \dots & \text{cov}(X_1, X_n) \\ \text{cov}(X_2, X_1) & \text{cov}(X_2, X_2) & \dots & \text{cov}(X_2, X_n) \\ \vdots & \vdots & \ddots & \vdots \\ \text{cov}(X_n, X_1) & \text{cov}(X_n, X_2) & \dots & \text{cov}(X_n, X_n) \end{bmatrix}$$

где  $\text{cov}(X_i, X_j)$  представляет собой ковариацию между  $X_i$  и  $X_j$ .

Ковариационная матрица является симметричной. В общем случае ковариация между двумя случайными переменными  $X_1$  и  $X_2$  является симметричной, поэтому  $\text{cov}(X_1, X_2) = \text{cov}(X_2, X_1)$ . Это свойство происходит из определения ковариации и не зависит от конкретных значений переменных. Таким образом, порядок указания переменных в ковариации не имеет значения, и результат будет одинаковым независимо от того, какую пару переменных мы рассматриваем.

Собственные векторы матрицы - это векторы, которые при умножении на эту матрицу остаются коллинеарными (линейно зависимыми) сами себе, изменяясь только в масштабе.

Визуальное представление о собственных векторах и значениях представлено на рисунке 7.10.

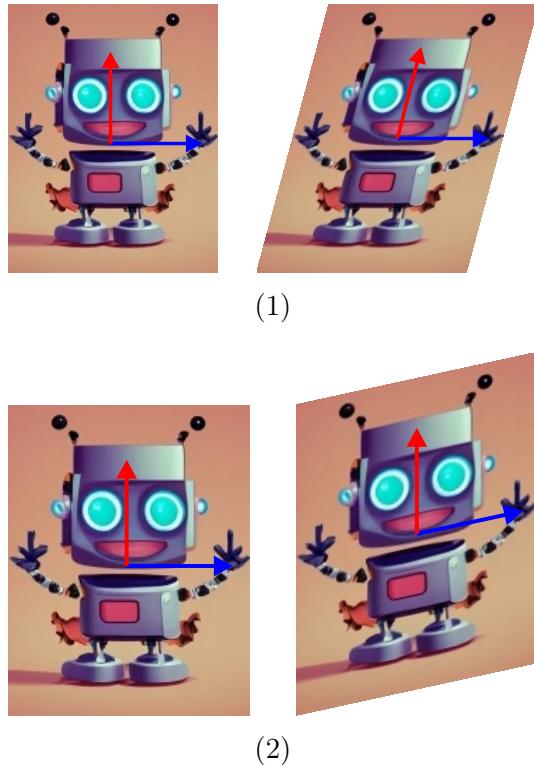


Рисунок 7.10: Визуальное представление о собственных векторах и значениях

Формально, собственные векторы определяются как ненулевые векторы  $\mathbf{v}$ , для которых выполняется следующее уравнение:

$$\mathbf{A} \cdot \mathbf{v} = \lambda \mathbf{v},$$

где  $\mathbf{A}$  - исходная матрица,  $\lambda$  - собственное значение (скаляр), а  $\mathbf{v}$  - собственный вектор.

Таким образом, умножение матрицы  $\mathbf{A}$  на собственный вектор  $\mathbf{v}$  приводит к получению нового вектора, который параллелен  $\mathbf{v}$  и масштабирован в  $\lambda$  раз. Собственные векторы матрицы играют важную роль в линейной алгебре. Они позволяют анализировать свойства и поведение матрицы, например, определять направления наибольшей и наименьшей изменчивости (главные компоненты) или решать системы линейных дифференциальных уравнений.

Собственные значения матрицы - это значения, которые удовлетворяют уравнению  $\mathbf{A} \cdot \mathbf{v} = \lambda \mathbf{v}$ , где  $\mathbf{A}$  - исходная матрица,  $\lambda$  - собственное значение (скаляр), а  $\mathbf{v}$  - собственный вектор.

Иными словами, собственные значения матрицы являются корнями уравнения  $\det(\mathbf{A} - \lambda \mathbf{I}) = 0$ , где  $\mathbf{I}$  - единичная матрица, а  $\det(\cdot)$  обозначает определитель матрицы.

Собственный вектор, обозначенный синим цветом на рисунке 7.10.1, остается параллельным самому себе при деформации или преобразовании. Это означает, что его направление не меняется, а только масштабируется (его длина изменяется). Такой вектор является собственным вектором преобразования, соответствующим определенному собственному значению  $\lambda$  (в данном случае  $\lambda = 1$ ), так как его направление сохраняется. Важно отметить, что любой вектор, параллельный синему собственному вектору, также будет собственным вектором с собственным значением  $\lambda$ .

Синий вектор меняет направление при деформации или преобразовании, в то время как красный вектор сохраняет свое направление. Поэтому красный вектор является собственным вектором, а синий - нет. Красный вектор не растягивается и не сжимается, его длина остается неизменной, поэтому соответствующее собственное значение равно единице, как показано на рисунке 7.10.2.

Собственные значения и собственные векторы играют важную роль в различных областях, таких как линейная алгебра, теория графов, физика, машинное обучение и другие. Они позволяют анализировать свойства и поведение матрицы, а также решать различные задачи, связанные с линейными операциями над данными.

Метод главных компонент и сингулярное разложение матриц (англ. Singular Value Decomposition, SVD) - это два связанных понятия в анализе данных и линейной алгебре. SVD - это разложение матрицы на три компонента:  $U$ ,  $\Sigma$  и  $V$ . SVD широко используется в линейной алгебре и анализе данных. SVD может быть применено к матрице данных, где каждая строка представляет собой наблюдение, а каждый столбец - признак. Разложение SVD может быть использовано для реализации PCA.

Связь между PCA и SVD состоит в том, что PCA может быть реализован с использованием SVD разложения. Если  $X$  - это матрица данных, то SVD разложение  $X$  даёт  $U$ ,  $\Sigma$  и  $V$ . Главные компоненты, полученные с помощью PCA, могут быть вычислены как линейная комбинация столбцов матрицы  $U$ , связанной с SVD разложением.  $\Sigma$  содержит сингулярные значения, которые представляют собой важность каждой главной компоненты.

Таким образом, SVD является математическим инструментом, который может быть использован для вычисления PCA и уменьшения размерности данных.

### 7.5.1 Визуальная демонстрация алгоритма

Объекты в наборах данных могут быть представлены разным количеством признаков. Мы можем визуализировать только те наборы данных, в которых не более 3 признаков (Рисунок 7.11).

Алгоритм PCA позволяет снизить количество признаков для визуализации или снижения размера датасета. В общем случае, количество признаков может быть снижено весьма существенно (например, для визуализации набора данных MNIST происходит снижение размерности с 784 до 2), однако в учебных целях мы будем преобразовывать данные в двумерном пространстве, произведя центрирование данных и смену осей.

Визуальная демонстрация алгоритма PCA может быть представлена в виде набора следующих шагов. Сначала мы читаем текущие данные и находим средние значения по каждому признаку, получая таким образом новый центр координат (Рисунок 7.12).

Затем мы перемещаем точку начала координат в этот новый усредненный центр нашего набора данных (Рисунок 7.13). Взаимное расположение объектов при этом сохраняется.

Затем мы находим линию (на рисунке обозначена красным пунктиром), проходящую через новый центр координат, сумма расстояний от точек данных до которой минимальна. Для

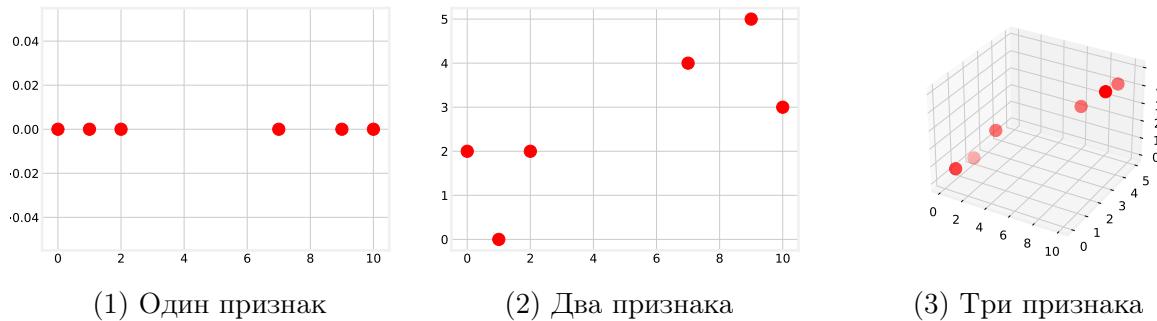


Рисунок 7.11: Разное количество признаков в датасете

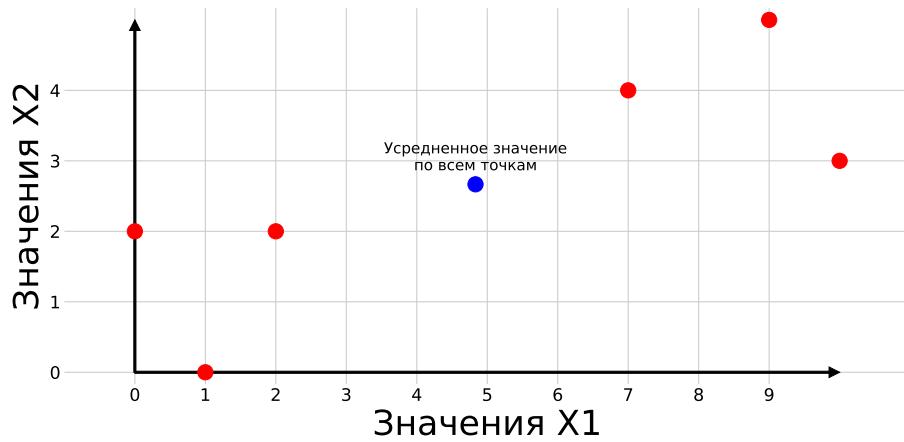


Рисунок 7.12: Нахождение среднего значения по признакам

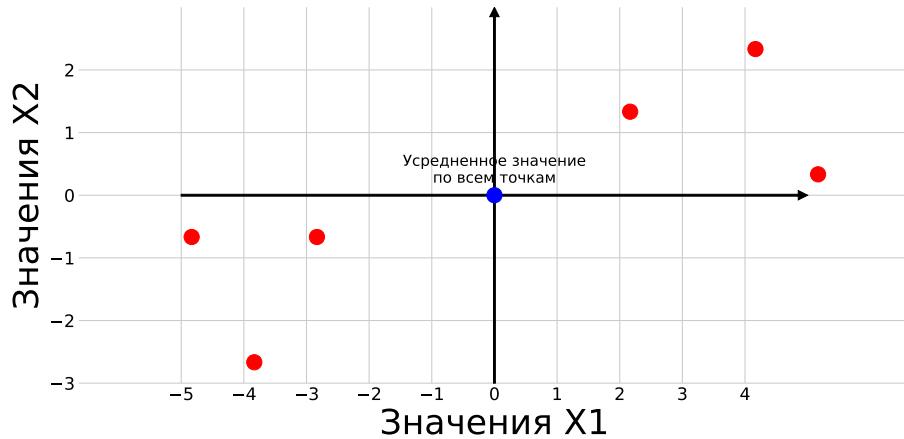


Рисунок 7.13: Центрирование данных

наглядности используется итеративный алгоритм (Рисунок 7.14), на практике используются другие техники, например, ковариационная матрица.

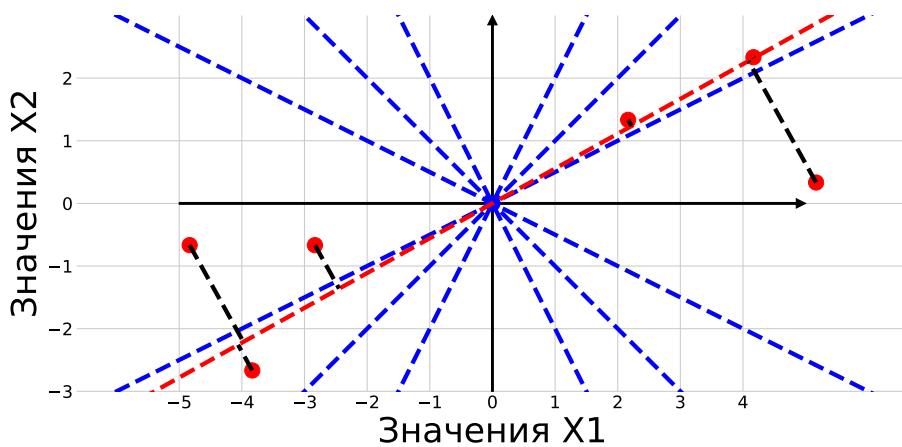


Рисунок 7.14: Нахождение линии, сумма расстояний от точек данных до которой минимальна

После этого, зная коэффициент наклона найденной прямой, находится линейная комбинация исходных признаков, которая определяет вектор, сонаправленный найденной линии, сумма расстояний от точек данных до которой минимальна. Нормировав все элементы линейной комбинации на длину этого вектора, мы получим собственный вектор первой главной компоненты длины 1 (на рисунке 7.15 синего цвета).

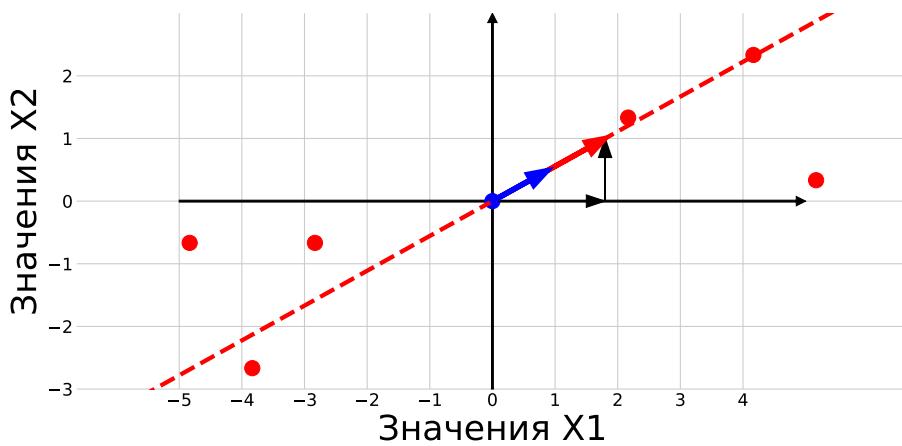


Рисунок 7.15: Нахождение линии, сумма расстояний от точек данных до которой минимальна

Также можно найти собственные значения главной компоненты. Каждое собственное значение соответствует одной главной компоненте и показывает, какую часть дисперсии в исходных данных объясняет соответствующая компонента. Сумма всех собственных значений равна общей дисперсии исходных данных.

Собственные значения используются для выбора количества главных компонент, которые нужно оставить после снижения размерности с помощью РСА. Чем больше собственное значение, тем больше доля дисперсии объясняется соответствующей главной компонентой. Обычно выбирают те главные компоненты, которые объясняют большую часть общей дисперсии (например, 95% или 99%) и игнорируют остальные компоненты, чтобы сократить размерность данных и сохранить наиболее информативные характеристики.

В наглядном примере формула для вычисления собственного значения главной компоненты равна:

$$\lambda = \frac{\sum_{i=1}^n \text{расстояние}^2 \text{ от } i\text{-го примера до линии главной компоненты}}{n - 1}$$

где  $n$  - количество объектов в наборе данных.

На практике формула для вычисления собственных значений в методе главных компонент (PCA) основана на ковариационной матрице данных.

Пусть  $C$  - ковариационная матрица размерности  $d \times d$  для исходных данных, где  $d$  - количество признаков.

Собственные значения  $\lambda$  можно получить как корни характеристического уравнения:

$$\det(C - \lambda I) = 0$$

где  $I$  - единичная матрица размерности  $d \times d$ .

Решив это уравнение, можно получить собственные значения  $\lambda_1, \lambda_2, \dots, \lambda_d$ .

Обратите внимание, что собственные значения обычно упорядочиваются по убыванию, что позволяет выбрать наиболее информативные главные компоненты, которые объясняют наибольшую часть дисперсии в данных.

Вторая главная компонента будет перпендикулярна первой. Аналогично, зная угловой коэффициент линии, перпендикулярной первой главной компоненте, мы можем найти линейные комбинации признаков, которые формируют вектор второй главной компоненты. Отнормировав все элементы линейной комбинации на длину полученного вектора, мы получим собственный вектор второй главной компоненты (Рисунок 7.16).

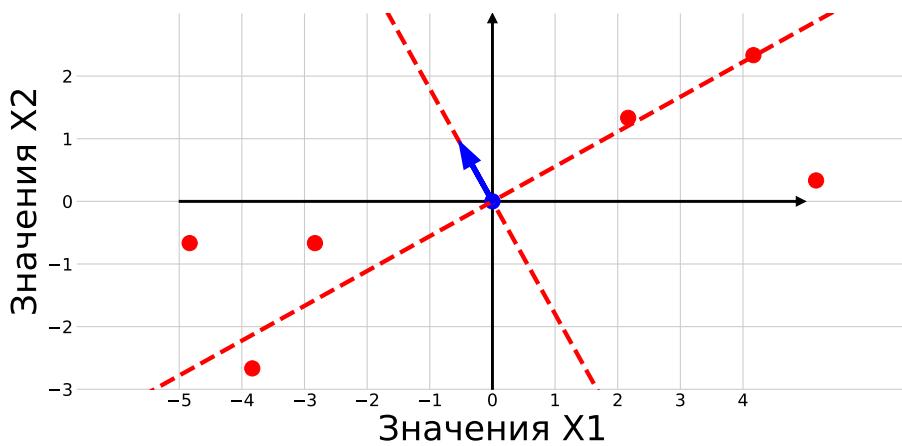


Рисунок 7.16: Нахождение собственного вектора второй главной компоненты

Если есть необходимость получить дальнейшие компоненты, процесс их поиска будет идти аналогичным образом.

В нашем случае мы остановимся после нахождения первых двух главных компонент. После вычисления этих компонент необходимо повернуть график таким образом, чтобы первая главная компонента стала новой осью обсцисс (осью X) (Рисунок 7.17). Соответственно, главные компоненты 1 и 2 становятся новыми осями координат.

Необходимо помнить, что каждая следующая главная компонента будет ортогональной предыдущим главным компонентам (Рисунок 7.18).

Также необходимо помнить, что чем выше собственные значения первых главных компонент, тем лучше они сохраняют информацию об исходном датасете (Рисунок 7.19). Если доля

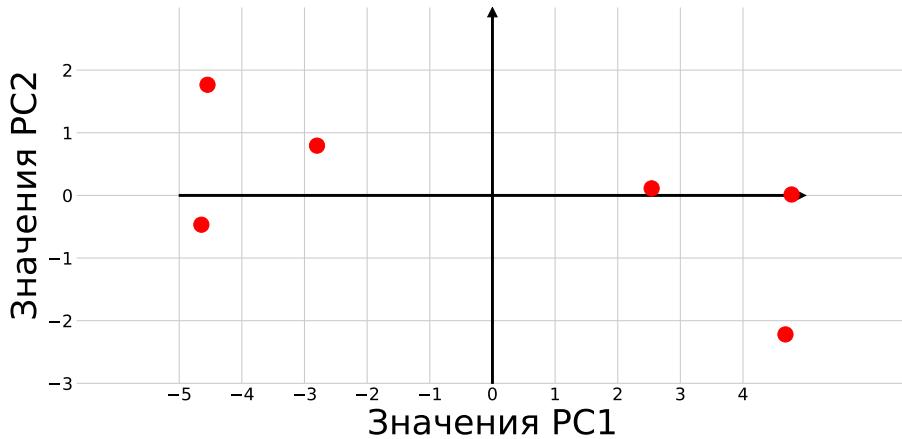


Рисунок 7.17: Поворот осей относительно начала координат

объясненной дисперсии среди первых компонент недостаточно высока, это может привести к снижения качества признакового описания, полученного в результате работы PCA.

### 7.5.2 Подготовка данных для алгоритма

Ниже представлен общий процесс подготовки данных для использования с методом главных компонент (PCA):

- Масштабирование данных: Рекомендуется масштабировать признаки входных данных перед применением PCA, особенно если признаки имеют различные шкалы или единицы измерения. Обычно используется стандартизация или нормализация данных для приведения их к общей шкале.
- Обработка категориальных переменных: Если у вас есть категориальные переменные в данных, требуется их преобразование в числовую формат. Вы можете использовать методы, такие как One-Hot Encoding или Label Encoding, чтобы закодировать категориальные переменные перед применением PCA.
- Обработка пропущенных значений: Если в ваших данных есть пропущенные значения, нужно решить, как с ними поступить. Вы можете удалить строки или столбцы с пропущенными значениями или заполнить их средними или медианными значениями в зависимости от контекста данных.

### 7.5.3 Процесс обучения

Шаги обучения метода главных компонент (PCA) с использованием сингулярного разложения (SVD) включают следующие действия:

1. Подготовка данных: Подготовьте данные, убедитесь, что они числовые и центрированы (имеют среднее значение равное нулю).
2. Вычисление матрицы ковариации: Вычислите матрицу ковариации для центрированных данных. Матрица ковариации имеет размерность  $d \times d$ , где  $d$  - количество признаков.

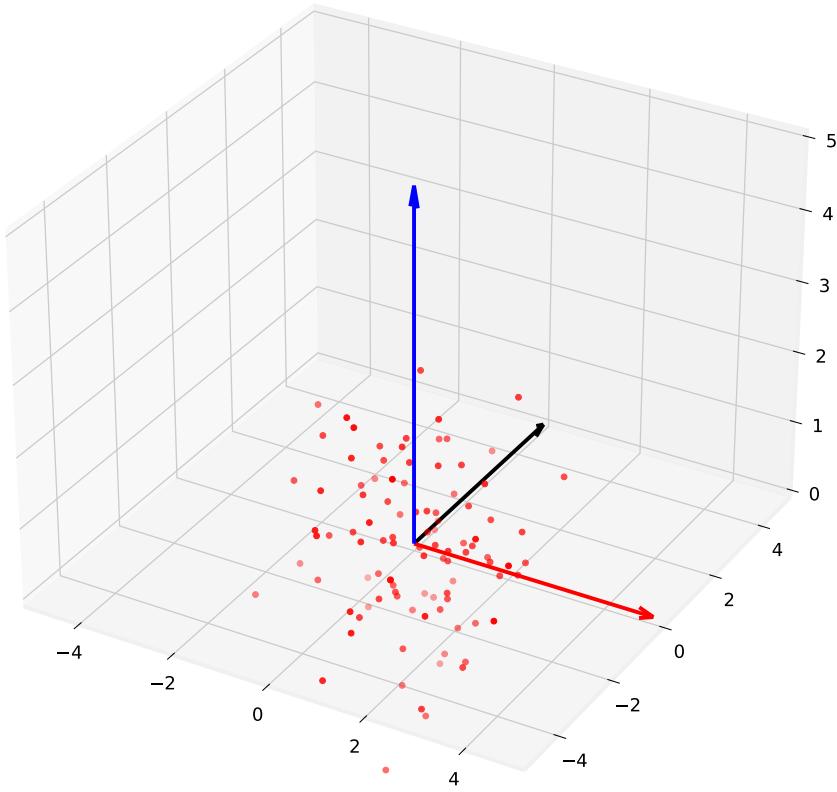


Рисунок 7.18: Ортогональность (перпендикулярность главных компонент)

3. SVD разложение: Примените сингулярное разложение (SVD) к матрице ковариации. SVD разложение представляет матрицу как произведение трех матриц:  $C = USV^T$ , где  $U$  - ортогональная матрица размерности  $d \times d$ ,  $S$  - диагональная матрица размерности  $d \times d$  с сингулярными значениями на диагонали, и  $V$  - ортогональная матрица размерности  $d \times d$ .
4. Выбор компонент: Определите, сколько главных компонент нужно оставить, выбирая сингулярные значения, которые объясняют большую часть общей дисперсии данных. Это можно сделать, например, с помощью критерия сохранения определенного процента дисперсии (например, 95% или 99%).
5. Проецирование на новое пространство: Проецируйте центрированные данные на пространство главных компонент. Для этого умножьте матрицу данных на матрицу главных компонент  $U_k$  размерности  $d \times k$ , где  $k$  - количество выбранных главных компонент.

#### 7.5.4 Оценка качества алгоритма

Оценка качества алгоритма РСА (метода главных компонент) может быть выполнена с использованием нескольких метрик и методов. Некоторые из них включают:

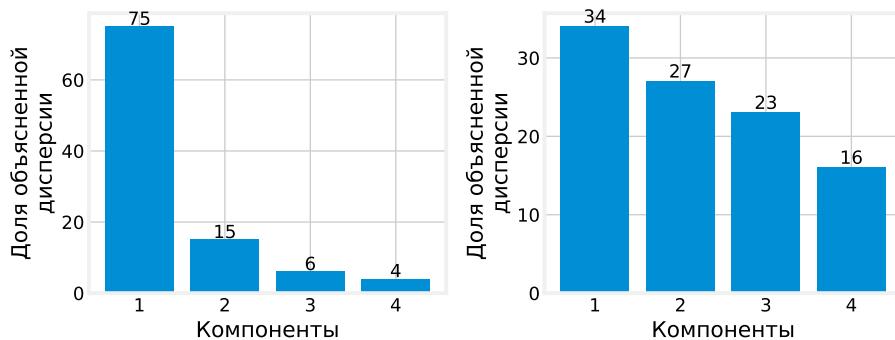


Рисунок 7.19: Высокая доля объясненной дисперсии среди первых главных компонент (график слева) и относительно низкая доля объясненной дисперсии среди первых главных компонент (график справа)

- Объясненная дисперсия: Оценка объясненной дисперсии позволяет определить, какую часть общей дисперсии данных объясняют выбранные главные компоненты. Эта метрика позволяет оценить, насколько успешно PCA сжимает информацию о данных.
- Кумулятивная объясненная дисперсия: Кумулятивная объясненная дисперсия показывает, сколько общей дисперсии данных объяснено суммарно выбранными главными компонентами. Она может быть полезна для определения оптимального числа главных компонент для использования.
- Восстановление данных: Оценка качества алгоритма PCA может быть выполнена путем восстановления данных из пространства главных компонент и сравнения восстановленных данных с исходными. Можно использовать различные метрики, такие как среднеквадратическая ошибка (MSE) или средняя абсолютная ошибка (MAE), чтобы оценить точность восстановления данных.
- Визуализация: Визуальное представление главных компонент и их влияния на данные может помочь в оценке качества алгоритма PCA. Построение графиков, например, графика объясненной дисперсии по числу главных компонент или графика сжатых данных в пространстве главных компонент, может помочь понять эффективность PCA.
- Задача: Наконец, оценка качества алгоритма PCA может быть выполнена с учетом конкретной задачи или цели. Например, в задаче классификации можно оценить, как PCA влияет на точность классификации модели, или какие признаки являются наиболее информативными после применения PCA.

### 7.5.5 Применение алгоритма

Алгоритм PCA имеет широкий спектр применений и может быть использован во многих областях. Вот некоторые из них:

- Снижение размерности данных: Одним из основных применений PCA является снижение размерности данных. Это позволяет представить многомерные данные в более компактной форме, удалив менее информативные признаки и выделив наиболее важные компоненты. Снижение размерности может быть полезно для улучшения эффективности вычислений, улучшения визуализации данных и борьбы с проклятием размерности.

- Визуализация данных: PCA может использоваться для визуализации данных в двух или трех измерениях. Применение PCA к исходным данным позволяет преобразовать их в новое пространство с меньшей размерностью, где можно визуализировать данные на плоскости или в пространстве. Это может помочь выявить структуру и зависимости между объектами данных.
- Удаление корреляций: PCA может использоваться для удаления или уменьшения корреляции между признаками в данных. Это может быть полезно в случаях, когда сильная корреляция между признаками мешает моделированию или приводит к мультиколлинеарности.
- Анализ главных компонент: PCA позволяет выделить главные компоненты, которые объясняют наибольшую долю дисперсии в данных. Это может помочь идентифицировать наиболее важные факторы или признаки, влияющие на данные, и понять их вклад в общую вариацию.
- Сжатие данных: PCA может использоваться для сжатия данных, сохраняя важную информацию и уменьшая объем памяти, необходимый для хранения данных. Это может быть полезно в задачах с ограниченными ресурсами или при передаче данных по сети.
- Предобработка данных: PCA может быть использован для предварительной обработки данных перед применением других алгоритмов машинного обучения. Он может помочь улучшить производительность моделей и избежать проблемы мультиколлинеарности.

Области применения PCA включают финансы, экономику, биоинформатику, обработку изображений, распознавание образов, геологию, маркетинг и многие другие.

### 7.5.6 Плюсы и минусы алгоритма

#### Плюсы

- Снижение размерности: Одним из основных преимуществ PCA является его способность снижать размерность данных. Он позволяет представить многомерные данные в пространстве меньшей размерности, сохранив при этом наибольшую долю информации. Это позволяет улучшить эффективность вычислений, уменьшить сложность моделей и улучшить визуализацию данных.
- Выделение наиболее информативных признаков: PCA позволяет определить наиболее важные компоненты данных, которые объясняют наибольшую долю вариации. Это помогает выделить наиболее информативные признаки и факторы, которые влияют на данные. Таким образом, PCA может быть использован для отбора признаков и уменьшения размерности без потери существенной информации.
- Устранение корреляций: PCA может использоваться для устранения или уменьшения корреляций между признаками. Это особенно полезно, когда сильная корреляция между признаками мешает моделированию или приводит к проблемам мультиколлинеарности.
- Визуализация данных: Применение PCA позволяет визуализировать данные в пространстве с меньшей размерностью. Это может помочь визуально исследовать структуру данных, выявить паттерны и зависимости, а также обнаружить аномалии или выбросы.

- Сжатие данных: PCA может быть использован для сжатия данных, уменьшая объем памяти, необходимый для хранения данных. При этом сохраняется основная структура и вариация данных. Сжатие данных может быть особенно полезным в задачах с ограниченными ресурсами или при передаче данных по сети.
- Предобработка данных: PCA может использоваться в качестве предварительной обработки данных перед применением других алгоритмов машинного обучения. Он может помочь улучшить производительность моделей, уменьшить влияние шума и избежать проблемы мультиколлинеарности.

### Минусы

- Потеря интерпретируемости: При снижении размерности данных с помощью PCA, исходные признаки объединяются в новые компоненты, которые являются линейными комбинациями исходных признаков. Это может привести к потере интерпретируемости, поскольку новые компоненты не всегда имеют прямую связь с исходными признаками.
- Зависимость от линейности: PCA предполагает линейные зависимости между признаками. Если данные имеют сложные нелинейные зависимости, PCA может быть неэффективным и не удастся захватить важную информацию, содержащуюся в данных.
- Чувствительность к выбросам: Алгоритм PCA может быть чувствительным к выбросам в данных. Выбросы могут сильно влиять на главные компоненты и искажать результаты анализа.
- Расчет вычислительно сложен: Вычисление собственных значений и собственных векторов для больших наборов данных может быть вычислительно сложной задачей. Это особенно актуально, когда количество признаков или объектов велико.
- Не учитывает контекст и доменные знания: PCA основывается на математических принципах и не учитывает контекст и доменные знания данных. Это может ограничивать его способность уловить специфические характеристики и зависимости в данных.

### 7.5.7 Реализация алгоритма в Python

PCA (Principal Component Analysis) в библиотеке scikit-learn реализована через класс PCA из модуля sklearn.decomposition. Основные параметры модели PCA:

- **n\_components:** количество главных компонент, которые требуется извлечь из данных. Это обязательный параметр.
- **svd\_solver:** метод, используемый для вычисления PCA. Может принимать значения «auto», «full», «arpack» или «randomized». По умолчанию используется «auto», который выбирает наиболее подходящий метод автоматически.

Класс PCA также имеет методы fit(X) для обучения модели на данных X, transform(X) для преобразования данных X с использованием изученных компонент, и fit\_transform(X) для комбинированного обучения и преобразования данных.

### 7.5.8 Вопросы для самопроверки

Какое утверждение о PCA верно (один ответ)?

1. PCA используется только для уменьшения размерности данных.
2. PCA используется только для визуализации данных.

3. РСА позволяет уменьшить размерность данных и сохранить наибольшую долю информации.
4. РСА применяется только к категориальным данным.

Правильные ответы: 3

Какие компоненты в РСА имеют наибольшее собственное значение?

1. Последние компоненты.
2. Произвольные компоненты.
3. Первые компоненты.
4. Компоненты с нулевым собственным значением.

Правильные ответы: 3

Какие преимуществами обладает РСА (шесть ответов)?

- Визуализация данных
- Выделение наиболее информативных признаков
- Зависимость от линейности
- Потеря интерпретируемости
- Предобработка данных
- Расчет вычислительно сложен
- Сжатие данных
- Снижение размерности
- Устранение корреляций
- Чувствительность к выбросам

Правильные ответы: 1, 2, 5, 7, 8, 9

### 7.5.9 Резюме по разделу

PCA (Principal Component Analysis) - это метод анализа данных, используемый для уменьшения размерности исходного набора данных. Основная идея РСА заключается в поиске новых независимых переменных, называемых главными компонентами, которые представляют наибольшую долю вариации в данных.

Основные преимущества РСА:

- Уменьшение размерности данных, что упрощает анализ и визуализацию.
- Сокращение шума и выбросов в данных.
- Сохранение наибольшей доли информации при уменьшении размерности данных.

РСА является мощным инструментом для анализа данных и широко применяется в области машинного обучения, статистики, визуализации данных и других областях, где требуется снижение размерности данных и выделение наиболее информативных признаков.

## 7.6 t-SNE

**Цель занятия:** ученик может применить алгоритм t-SNE для решения задач снижения размерности на подготовленных и неподготовленных данных.

**План занятия:**

- Визуальная демонстрация алгоритма
- Подготовка данных для алгоритма
- Процесс обучения
- Оценка качества алгоритма
- Применение алгоритма
- Плюсы и минусы алгоритма
- Реализация алгоритма в Python

t-SNE (t-Distributed Stochastic Neighbor Embedding) — это алгоритм машинного обучения, используемый для визуализации и снижения размерности данных. Он был разработан для отображения сложных многомерных данных в двух- или трехмерное пространство, сохраняя при этом сходства между точками.

Основная идея t-SNE заключается в том, чтобы представить исходные данные в новом пространстве, где близкие объекты остаются близкими, а далекие объекты разделяются. Алгоритм строит вероятностное распределение для пар объектов в исходном пространстве и в новом пространстве таким образом, чтобы минимизировать разницу между ними. Также t-SNE позволяет учитывать локальные и глобальные структуры данных.

t-SNE широко используется в визуализации данных, особенно при работе с высокоразмерными и сложными наборами данных, такими как изображения или тексты. Он помогает обнаруживать скрытые паттерны, кластеры и взаимосвязи между объектами в данных.

Перед тем, как рассмотреть алгоритм t-SNE, необходимо освежить в памяти некоторые понятия. Нормальное распределение, также известное как распределение Гаусса, является одним из наиболее распространенных и важных вероятностных распределений. Оно характеризуется плотностью вероятности, которая имеет форму колокола или симметричного колоколообразного графика (Рисунок 7.20).



Рисунок 7.20: График функции плотности вероятности нормального распределения

Формула плотности вероятности нормального распределения выглядит следующим образом:

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} \cdot e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

где:  $x$  - случайная переменная,  $\mu$  - математическое ожидание (среднее значение) распределения,  $\sigma^2$  - дисперсия (квадрат стандартного отклонения) распределения.

В этой формуле параметры  $\mu$  и  $\sigma^2$  определяют положение и форму колокола нормального распределения. Математическое ожидание  $\mu$  указывает на центр колокола, а дисперсия  $\sigma^2$  определяет его ширину. Распределение Стьюдента ( $t$ -распределение) - это вероятностное распределение, которое широко используется при оценке параметров и проведении статистических тестов в малых выборках, когда исходная генеральная совокупность имеет нормальное распределение, но значение дисперсии неизвестно.

Распределение Стьюдента с параметром  $\nu$ , обозначаемое как  $t(\nu)$ , имеет форму колокола с симметричным графиком вокруг нуля. Число степеней свободы  $\nu$  определяет форму распределения и влияет на его хвостатость. Чем больше  $\nu$ , тем ближе распределение Стьюдента приближается к стандартному нормальному распределению. (Рисунок 7.21).



Рисунок 7.21: График функции плотности вероятности распределения Стьюдента

Формула плотности вероятности распределения Стьюдента выглядит следующим образом:

$$f(x) = \frac{\Gamma\left(\frac{\nu+1}{2}\right)}{\sqrt{\nu\pi\Gamma\left(\frac{\nu}{2}\right)}} \left(1 + \frac{x^2}{\nu}\right)^{-\frac{\nu+1}{2}}$$

где:  $x$  - случайная переменная,  $\nu$  - число степеней свободы (degrees of freedom) распределения,  $\Gamma$  - функция Гамма, которая вычисляется для соответствующих аргументов.

Обратите внимание, что для вычисления плотности вероятности распределения Стьюдента требуется использование функции Гамма, которая может быть определена отдельно.

Гауссовское ядро (или ядро Гаусса) - это одно из распространенных ядерных функций, используемых в алгоритмах машинного обучения и обработки сигналов. Оно часто применяется в задачах сглаживания и фильтрации данных. Гауссовское ядро представляет собой функцию, которая зависит от расстояния между двумя точками и имеет форму колокола с пиком в нуле. Оно симметрично и уменьшается с расстоянием от пика. Гауссовское ядро используется для вычисления весовых коэффициентов в методе ядерного сглаживания (kernel smoothing). В этом методе, каждая точка данных взвешивается с помощью функции

гауссовского ядра в зависимости от её расстояния от интересующей нас точки. Ближние точки получают больший вес, тогда как дальние точки получают меньший вес (Рисунок 7.22). Формула Гауссовского ядра выглядит следующим образом:

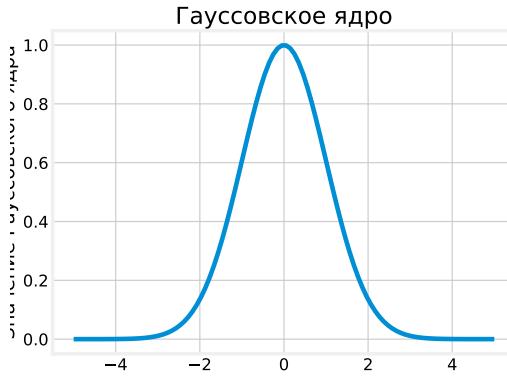


Рисунок 7.22: Гауссовское ядро

$$K(x, x') = \exp\left(-\frac{\|x - x'\|^2}{2\sigma^2}\right)$$

где: -  $x$  и  $x'$  - две точки, между которыми мы вычисляем ядро, -  $\|x - x'\|$  - расстояние между точками  $x$  и  $x'$ , -  $\sigma$  - параметр ширины ядра (стандартное отклонение).

В формуле используется евклидово расстояние между точками  $x$  и  $x'$ , и оно возведено в квадрат. Параметр  $\sigma$  определяет ширину ядра и влияет на «размытость» и «распределение массы» вокруг каждой точки.

Гауссовское ядро широко применяется в алгоритмах машинного обучения, таких как метод опорных векторов (SVM) и метод гауссовых процессов (Gaussian Processes), а также в различных методах сглаживания и фильтрации данных. Дивергенция Кульбака-Лейблера (Kullback-Leibler divergence), также известная как относительная энтропия, является мерой различия между двумя вероятностными распределениями. Она широко применяется в области информационной теории и статистики для измерения расстояния между двумя распределениями. Дивергенция Кульбака-Лейблера между двумя вероятностными распределениями  $P$  и  $Q$  измеряет, насколько сильно распределение  $P$  отличается от распределения  $Q$ . Она не является метрикой расстояния, так как она не обладает свойством симметричности, то есть  $KL$ -дивергенция между распределением  $P$  и распределением  $Q$  не равна  $KL$ -дивергенции между распределением  $Q$  и распределением  $P$ .

Формула дивергенции Кульбака-Лейблера выглядит следующим образом:

$$D_{KL}(P||Q) = \sum_i P(i) \log\left(\frac{P(i)}{Q(i)}\right)$$

где: -  $D_{KL}$  - дивергенция Кульбака-Лейблера, -  $P(i)$  и  $Q(i)$  - вероятности событий  $i$  в распределениях  $P$  и  $Q$  соответственно, -  $\log$  - натуральный логарифм.

Обратите внимание, что дивергенция Кульбака-Лейблера может быть определена только в случае, когда все значения  $P(i)$  равны нулю, если  $Q(i)$  равно нулю.

Дивергенция Кульбака-Лейблера является положительной и неотрицательной, и равна нулю только в случае, когда два распределения  $P$  и  $Q$  совпадают.

### 7.6.1 Визуальная демонстрация алгоритма

Визуальная демонстрация алгоритма t-SNE представлена на рисунке ???. Необходимо загрузить в память исходную выборку с большим количеством признаков и случайным образом сгенерировать её отображение в признаковое пространство меньшей размерности (Рисунок 7.23).

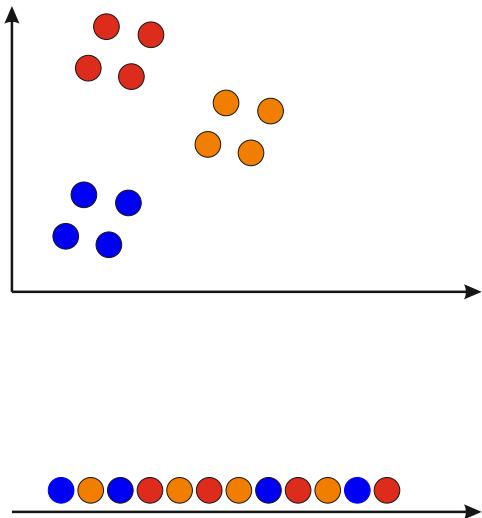


Рисунок 7.23

Основная идея алгоритма t-SNE заключается в том, чтобы сохранить близость объектов из исходного пространства в результатеющем пространстве (Рисунок 7.24).

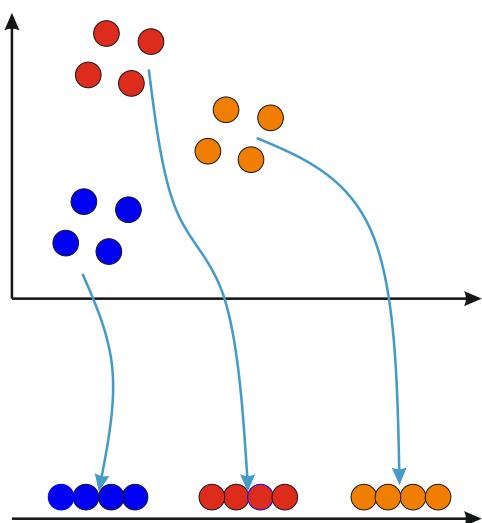


Рисунок 7.24

Сначала вычисляется попарная сходственность (affinity) между всеми парами объектов в исходном пространстве. Обычно используется гауссово ядро (гауссовое распределение) для оценки сходства между объектами. Сходство выражается числовыми значениями, которые показывают степень близости между объектами. Затем вычисляются условные вероятности сходства между парами объектов на основе попарных сходств. Эти вероятности вычисляются как отношение попарных сходств к сумме всех сходств, взятых с учетом попарных расстояний между объектами (Рисунок 7.25).

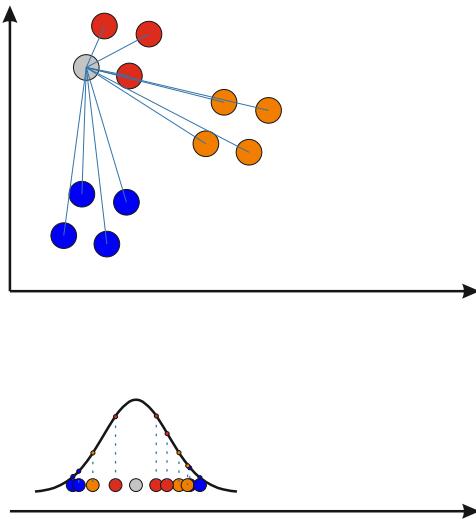


Рисунок 7.25

Симметричность расстояний от точки  $i$  до точки  $j$  и перплексия - два важных фактора для правильного функционирования алгоритма t-SNE и получения высококачественных вложений. Симметричность расстояний для метрики означает, что расстояние от точки  $i$  до точки  $j$  и наоборот равны. Изначально в дивергенции Кульбака-Лейблера это не так. В алгоритме t-SNE производится усреднение расстояний от точки  $i$  до точки  $j$  и наоборот (то есть симметризация расстояний) для достижения более устойчивых и сбалансированных вложений в новом пространстве. Перплексия (perplexity) в теории информации - это мера сложности или неопределенности распределения вероятностей. Она используется в различных контекстах, включая машинное обучение и статистику.

Математически, перплексия определяется как экспонента энтропии распределения вероятностей. Для дискретного случая, перплексия вычисляется следующим образом:

$$\text{Perplexity}(P) = 2^{-\sum_i P(i) \log_2 P(i)}$$

где  $P$  - распределение вероятностей.

Для непрерывного случая, перплексия может быть определена как:

$$\text{Perplexity}(p(x)) = \exp \left( - \int p(x) \log_2 p(x) dx \right)$$

где  $p(x)$  - плотность вероятности.

Перплексия в алгоритме t-SNE используется для контроля числа ближайших соседей, которые учитываются при вычислении условных вероятностей сходства. Высокое значение перплексии увеличивает влияние ближайших соседей, тогда как низкое значение перплексии распространяет влияние на более широкий набор объектов. Выбор оптимального значения перплексии зависит от особенностей данных и требуемой структуры вложений. Подбор правильной перплексии позволяет достичь более точного отображения локальных сходств и сохранения структуры данных в новом пространстве (Рисунок 7.26).

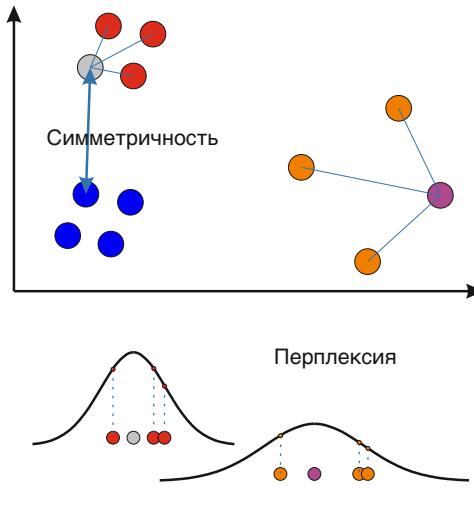


Рисунок 7.26

Симметричность расстояний и перплексия взаимодействуют, чтобы учесть локальные отношения между объектами и сохранить структуру данных в новом пространстве.

На каждой итерации алгоритма t-SNE вычисляется условная вероятность сходства объектов. Для каждой пары объектов  $i$  и  $j$  вычисляется условная вероятность  $p_{ij}$ , которая отражает вероятность выбрать объект  $j$  как соседа объекта  $i$  при условии их исходного расположения в пространстве высокой размерности. Затем эти условные вероятности  $p_{ij}$  сохраняются в матрице  $P$ , где элемент  $P_{ij}$  представляет собой условную вероятность сходства между объектами  $i$  и  $j$ .

Также, на следующем шаге текущей итерации алгоритма t-SNE, вычисляется условная вероятность в новом пространстве низкой размерности. Для каждой пары объектов в новом пространстве  $i'$  и  $j'$ , вычисляется условная вероятность  $q_{ij}$ , которая отражает вероятность выбрать объект  $j'$  как соседа объекта  $i'$ . Эти условные вероятности  $q_{ij}$  также сохраняются в матрице  $Q$ . Сохранение вероятностей сходства в матрицах  $P$  и  $Q$  является важным для дальнейшего вычисления расстояний и оптимизации вложений в пространстве низкой размерности.

Далее происходит оптимизация расположения объектов в результирующем пространстве. Алгоритм стремится минимизировать расхождение между условными вероятностями сходства в исходном пространстве и результирующем пространстве с помощью дивергенции Кульбака-Лейблера. Оптимизация выполняется с использованием метода градиентного спуска. В каждой итерации оптимизации происходит обновление вложений (координат) объ-

ектов в результирующем пространстве. Обновление выполняется путем расчета градиента и изменения координат объектов в направлении, которое минимизирует расхождение между вероятностями сходства объектов, сохраненными в соответствующих матрицах. Алгоритм повторяется до достижения сходимости. Обычно определенное число итераций задается заранее или алгоритм останавливается, когда изменение вложений становится незначительным. В результате работы алгоритма t-SNE получается новое представление данных в пространстве меньшей размерности, которое учитывает локальные сходства между объектами. Это позволяет визуализировать данные и обнаруживать структуры и паттерны, которые могут быть неочевидны в исходном пространстве высокой размерности (Рисунок 7.27).

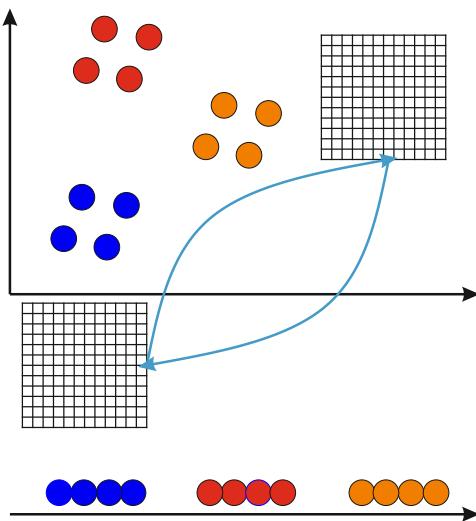


Рисунок 7.27

## 7.6.2 Процесс обучения

Шаги алгоритма t-SNE:

1. Вычисление схожести: Сначала алгоритм вычисляет меру схожести между парами объектов в исходном пространстве. Обычно это делается с использованием гауссовского ядра или расстояния между точками, такого как евклидово расстояние.
2. Создание вероятностного распределения: Для каждой точки в исходном пространстве алгоритм вычисляет вероятностное распределение, отражающее схожесть между этой точкой и другими точками. Более похожим точкам присваивается более высокая вероятность.
3. Вычисление схожести в пространстве назначения: Затем алгоритм переходит в пространство назначения (обычно двух- или трехмерное), где он создает аналогичное вероятностное распределение. Он старается разместить точки таким образом, чтобы более похожие точки были ближе друг к другу.
4. Минимизация дивергенции Кульбака-Лейблера: Целью t-SNE является минимизация дивергенции Кульбака-Лейблера между вероятностными распределениями в исходном

пространстве и пространстве назначения. Это достигается путем настройки позиций точек в пространстве назначения таким образом, чтобы минимизировать разницу между этими распределениями.

5. Градиентный спуск: Для минимизации дивергенции Кульбака-Лейблера алгоритм использует градиентный спуск. Он вычисляет градиент функционала ошибки и обновляет позиции точек, чтобы уменьшить ошибку. Обновление происходит итеративно, пока не будет достигнута сходимость.

### 7.6.3 Оценка качества алгоритма

Для оценки качества работы алгоритма t-SNE можно использовать несколько метрик. Вот некоторые из них:

- Визуализация: Одним из наиболее популярных способов оценки качества t-SNE является визуальный анализ результатов. Просмотрите полученное низкоразмерное представление данных и оцените, насколько хорошо алгоритм сохраняет структуру и относительные расстояния между объектами. Убедитесь, что близкие объекты находятся близко друг к другу, а разные кластеры отделены друг от друга.
- Поддержка кластеризации: Если у вас есть информация о настоящих кластерах в данных, вы можете оценить, насколько хорошо t-SNE справляется с их выделением. Используйте метрики, такие как Adjusted Rand Index (ARI), Normalized Mutual Information (NMI) или Silhouette Score, чтобы сравнить полученные кластеры с эталонными.
- Сохранение расстояний: t-SNE должен сохранять относительные расстояния между объектами в исходном пространстве при их проецировании на низкоразмерное пространство. Вы можете вычислить попарные расстояния между объектами в исходном и низкоразмерном пространствах и сравнить их. Можно использовать метрики, такие как Pearson correlation coefficient или Spearman rank correlation coefficient для оценки сходства между расстояниями.
- Stochastic Neighbor Embedding (SNE) cost function: t-SNE оптимизирует функцию стоимости, основанную на SNE. Вы можете отслеживать значения функции стоимости на каждой итерации и проверить, сходится ли алгоритм. Если значения функции стоимости продолжают уменьшаться, это может быть индикатором успешной работы алгоритма.

Важно отметить, что t-SNE является эвристическим алгоритмом, и не существует универсальных метрик, которые идеально оценивают его качество. Оценка качества t-SNE должна основываться на комбинации различных метрик и визуальной интерпретации результатов.

### 7.6.4 Применение алгоритма

Алгоритм t-SNE является мощным инструментом для визуализации исходных данных в низкоразмерном пространстве. Он может использоваться в различных областях, где требуется визуализация и анализ многомерных данных. Вот некоторые примеры применения алгоритма t-SNE:

- Визуализация данных: Основное применение t-SNE заключается в визуализации сложных многомерных данных в двух или трех измерениях. Это позволяет исследовать и понять структуру данных, выявить кластеры или группы объектов, обнаружить выбросы

или аномалии и обнаружить скрытые паттерны или зависимости. Примеры включают визуализацию геномных данных, изображений, текстовых данных и т.д.

- Кластеризация: t-SNE может использоваться для кластеризации данных на основе их визуализации в низкоразмерном пространстве. После проецирования данных на плоскость с помощью t-SNE, можно применить алгоритмы кластеризации, такие как k-means или DBSCAN, для выделения кластеров и их анализа.
- Обнаружение аномалий: t-SNE может помочь в обнаружении аномалий или выбросов в данных. Объекты, которые отображаются далеко от основной структуры данных в низкоразмерном пространстве, могут считаться потенциальными аномалиями или интересными случаями, которые заслуживают дополнительного исследования.
- Предварительная обработка данных: t-SNE может использоваться как этап предварительной обработки данных для последующего применения других алгоритмов машинного обучения. Проецирование данных на низкоразмерное пространство с помощью t-SNE может улучшить эффективность и качество работы других алгоритмов, таких как классификация, регрессия или кластеризация.
- 

Важно отметить, что t-SNE является вычислительно сложным алгоритмом, особенно для больших наборов данных. Поэтому для его применения к большим данным может потребоваться распараллеливание вычислений или использование других методов.

### 7.6.5 Плюсы и минусы алгоритма

**Плюсы** Алгоритм t-SNE имеет несколько преимуществ, которые делают его популярным инструментом для визуализации и анализа данных. Вот некоторые из плюсов алгоритма t-SNE:

- Сохранение локальной структуры: t-SNE стремится сохранить относительные расстояния между близкими объектами в исходном пространстве. Это означает, что объекты, которые находятся близко друг к другу в исходных данных, будут отображены близко друг к другу в низкоразмерном пространстве. Это позволяет сохранить локальную структуру данных и обнаружить кластеры или группы объектов.
- Устойчивость к выбросам: t-SNE обычно хорошо справляется с выбросами или аномалиями в данных. Из-за своего вероятностного подхода и использования тяжелых хвостов распределения t-Student, t-SNE не так сильно подвержен влиянию выбросов и может помочь обнаружить их.
- Визуальная интерпретация: Одним из основных преимуществ t-SNE является его способность визуализировать высокоразмерные данные в двух или трех измерениях. Это позволяет исследовать данные, выявлять паттерны и структуры, а также делать выводы и принимать решения на основе визуальной интерпретации результатов.

**Минусы** Наряду с преимуществами, алгоритм t-SNE также имеет некоторые ограничения и потенциальные недостатки. Вот некоторые из них:

- Вычислительная сложность: Алгоритм t-SNE требует вычисления попарных расстояний между всеми парами точек в исходном пространстве. Это делает его вычислительно требовательным, особенно для больших наборов данных. Время выполнения может значительно возрастать с увеличением размерности исходных данных или числа объектов.

- Недетерминированность: t-SNE является стохастическим алгоритмом, что означает, что результаты могут немного различаться при каждом запуске. Это может затруднить воспроизводимость результатов и требует выполнения нескольких запусков для проверки стабильности и надежности результатов.
- Потеря глобальной структуры: t-SNE сконцентрирован на сохранении локальной структуры данных и обнаружении кластеров. Однако, из-за своей природы, алгоритм может снижать различия между удаленными объектами в низкоразмерном пространстве, что может привести к потере глобальной структуры данных.

### 7.6.6 Реализация алгоритма в Python

В библиотеке scikit-learn, алгоритм t-SNE (t-Distributed Stochastic Neighbor Embedding) реализован через класс TSNE из модуля sklearn.manifold. Описание основных параметров класса TSNE:

- **n\_components:** количество компонентов в низкоразмерном пространстве, в которое будет проецироваться исходное пространство данных. Обычно это 2 или 3 для визуализации. Это обязательный параметр.
- **perplexity:** мера сложности глобальной структуры данных. Определяет баланс между сохранением локальной и глобальной структуры. Значения perplexity должны быть в диапазоне от 5 до 50, но часто используется значение около 30.
- **learning\_rate:** скорость обучения алгоритма. Определяет шаг изменения в низкоразмерном пространстве. Маленькое значение learning\_rate может привести к более долгой сходимости, но лучшей качеству визуализации. Значения learning\_rate обычно варьируются от 10 до 1000.
- **n\_iter:** количество итераций оптимизации алгоритма. Определяет количество шагов, которые алгоритм выполняет для достижения оптимального результата. Значение по умолчанию - 1000.

В классе TSNE также имеются методы **fit(X)** для обучения модели на данных X и **transform(X)** для преобразования данных X в низкоразмерное пространство с использованием изученных компонент.

### 7.6.7 Вопросы для самопроверки

Какое утверждение о t-SNE верно (один ответ)?

1. t-SNE используется только для уменьшения размерности данных.
2. t-SNE используется только для визуализации данных.
3. t-SNE основывается на вероятностной модели, которая стремится сохранить схожесть точек в исходном пространстве при их проецировании в низкоразмерное пространство.
4. t-SNE применяется для задач классификации.

Правильные ответы: 3

Какие параметры нужно установить для работы t-SNE (два ответа)?

1. Число соседей исходных данных
2. Размерность входных данных

3. Число итераций алгоритма
4. perplexity и learning rate

Правильные ответы: 3, 4

Какие преимуществами обладает t-SNE (три ответа)?

- Визуальная интерпретация
- Вычислительная сложность
- Недетерминированность
- Потеря глобальной структуры
- Сохранение локальной структуры
- Устойчивость к выбросам

Правильные ответы: 1, 5, 6

### 7.6.8 Резюме по разделу

t-SNE (t-Distributed Stochastic Neighbor Embedding) - это алгоритм снижения размерности данных, который часто используется для отображения высокоразмерных данных на двух- или трехмерное пространство. Он широко применяется для анализа и визуализации сложных наборов данных, таких как текстовые, аудио, графовые и другие.

Основная идея алгоритма t-SNE заключается в том, чтобы сохранить близость объектов из исходного пространства в результирующем пространстве. Алгоритм строит вероятностную модель, где объекты в исходном пространстве и в пространстве визуализации представлены вероятностными распределениями. Он пытается минимизировать расхождение между этими распределениями, чтобы сохранить связи и соседство между объектами.

Для работы алгоритма t-SNE необходимо настроить параметры, такие как perplexity (параметр, определяющий баланс между сохранением локальной и глобальной структуры данных) и learning rate (параметр, регулирующий скорость обучения). Оптимальный выбор этих параметров может существенно влиять на качество визуализации.

## 7.7 Резюме по модулю

Обучение без учителя (англ. Unsupervised learning) - это подраздел машинного обучения, в котором модель обучается на неразмеченных данных без наличия явно заданных целевых переменных или меток. В отличие от обучения с учителем, где модель обучается на парах «объект-ответ», в задачах обучения без учителя нет явно определенных целей или правильных ответов. Основная цель обучения без учителя состоит в извлечении скрытых структур, паттернов или информации из неразмеченных данных.

Основные задачи обучения без учителя включают:

- Кластеризация: Задача заключается в разделении набора данных на группы или кластеры, где объекты внутри каждого кластера схожи между собой, а объекты из разных кластеров различаются. Кластеризация помогает выявить скрытую структуру в данных и понять, как объекты могут быть группированы на основе их сходства.
- Снижение размерности: Задача состоит в уменьшении размерности данных путем проецирования их на пространство меньшей размерности. Целью является сохранение важных характеристик и структуры данных, одновременно устранив шум и избыточность. Понижение размерности упрощает визуализацию и анализ данных, а также может улучшить производительность моделей машинного обучения.

Основные алгоритмы кластеризации:

- К-средних (K-means): Это один из наиболее популярных и простых алгоритмов кластеризации. Он разбивает набор данных на K кластеров, где каждый объект присваивается к ближайшему центроиду. Центроиды пересчитываются на каждой итерации до достижения сходимости.
- Иерархическая кластеризация: Этот алгоритм строит иерархическую структуру кластеров, которая может быть представлена в виде дендрограммы. Он может быть агломеративным, начиная с каждого объекта в отдельном кластере и объединяя их постепенно, или дивизивным, начиная с одного кластера и разделяя его на подкластеры.
- DBSCAN (Density-Based Spatial Clustering of Applications with Noise): Этот алгоритм основан на плотности данных. Он определяет кластеры, исходя из плотности объектов в их окрестности. DBSCAN может обнаруживать кластеры произвольной формы и имеет возможность обнаруживать выбросы.
- GMM (Gaussian Mixture Model): Это вероятностная модель, которая моделирует данные как смесь нескольких гауссовых распределений. GMM предполагает, что каждый кластер имеет свое гауссово распределение, и объекты внутри кластера генерируются из этого распределения. В рамках этого курса мы его не рассматриваем.

Основные алгоритмы снижения размерности:

- PCA (Principal Component Analysis): Это один из наиболее распространенных алгоритмов снижения размерности. PCA находит линейные комбинации исходных признаков, называемые главными компонентами, которые объясняют наибольшую дисперсию в данных. После этого можно выбрать первые k главных компонент для сокращения размерности данных.
- t-SNE (t-Distributed Stochastic Neighbor Embedding): Этот алгоритм используется для визуализации и снижения размерности данных. Он стремится сохранить локальные сходства объектов, перенося их на пространство меньшей размерности. t-SNE обычно

хорошо работает для сохранения глобальных структур данных, но может иметь проблемы с сохранением глобальных расстояний.

- UMAP (Uniform Manifold Approximation and Projection): Этот алгоритм является относительно новым методом снижения размерности, который также используется для визуализации данных. UMAP стремится сохранить глобальную структуру данных и их локальные сходства. Он основан на построении графа соседей и оптимизации распределения объектов на меньшую размерность с сохранением сходства соседей. В рамках этого курса мы его не рассматриваем.