



Бернгардт О. И.

# АНАЛИЗ ДАННЫХ

на Python



Конспект лекций

Анне, Яну, и Софье с благодарностью  
за постоянную помощь и поддержку

Проект выполнен при поддержке Фонда Владимира  
Потанина, грантовый конкурс для преподавателей  
магистратуры 2025-2026.



# Оглавление

1.Введение.....	6
1.1.Виды анализа данных.....	6
1.1.1.Описательный анализ.....	6
1.1.1.2. Типизация данных.....	7
1.1.1.2. Базовые методы агрегации данных.....	9
1.1.1.3.Методы визуализации.....	10
1.1.2.Диагностический анализ.....	11
1.1.3.Прогностический анализ.....	11
1.1.4.Рекомендательный анализ.....	11
2.Вероятность.....	12
2.1. Определение.....	12
2.2. Связанные события. Полная и условная вероятность.....	12
3.Статистические распределения.....	14
3.1. Дискретные распределения.....	14
3.1.1.Распределение Бернулли.....	14
3.1.2.Биномиальное распределение.....	15
3.1.3.Распределение Пуассона.....	16
3.2.Непрерывные распределения.....	17
3.2.1.Равномерное распределение.....	17
3.2.2.Нормальное распределение.....	19
3.2.3.Распределение хи-квадрат.....	20
3.3.Центральная предельная теорема.....	21
3.4.Ядерное сглаживание.....	22
3.5.Построение распределений в библиотеке scipy.....	23
3.5.1. Статистики распределений.....	23
3.5.2. Доверительные интервалы и квантили. Уровень значимости.....	25
4.Корреляционный анализ.....	26
4.1.Связь числовых величин.....	26
4.1.1.Корреляция Пирсона.....	26
4.1.2.Корреляция Спирмена.....	26
4.2.Связь номинальных величин.....	27
4.2.1.Коэффициент V Крамера.....	28
4.2.2.Индекс Рэнда.....	28
4.2.3.Скорректированный индекс Рэнда.....	29
4.3.Связь порядковых и числовых(или интервальных) величин.....	30
4.4.Связь номинальных и числовых величин.....	30
5.Кластеризация.....	31
5.1.Введение.....	31
5.2.Плоские алгоритмы.....	31
5.2.1.Метрические методы кластеризации.....	31
5.2.1.1.DBSCAN.....	31
5.2.1.2.Метрики качества основанные на расстоянии.....	33
5.2.2.Модельные методы кластеризации.....	35
5.2.2.1.Гауссова смесь.....	35
5.2.2.2.Метрики качества основанные на вероятности.....	37
5.3.Иерархические алгоритмы.....	38

5.3.1.Агglomerативная кластеризация.....	38
5.3.2.Дивизивная кластеризации.....	39
5.3.3.Метрики качества.....	41
5.4.Смешанные алгоритмы.....	41
6.Проверка статистических гипотез.....	43
6.1.Введение.....	43
6.2.Параметрические критерии.....	44
6.2.1.Одновыборочные критерии.....	44
6.2.1.1.Z-критерий для среднего.....	44
6.2.1.2.T-критерий для среднего.....	45
6.2.1.3.Z-критерий для доли.....	46
6.2.2.Двухвыборочные критерии для несвязанных выборок.....	47
6.2.2.1.Z-критерий для среднего.....	47
6.2.2.2.T-критерий Уэлча для среднего.....	48
6.2.2.3.Z-критерий для доли.....	49
6.2.3.Двухвыборочные критерии для связанных выборок.....	50
6.2.3.1.T-критерий для средних.....	50
6.2.3.2.Z-критерий для доли.....	51
6.3.Непараметрические критерии.....	52
6.3.1. Одновыборочный критерий Уилкоксона.....	52
6.3.2. Двухвыборочный критерий Мана-Уитни (несвязанные выборки).....	53
6.3.3. Двухвыборочный критерий Уилкоксона (связанные выборки).....	54
6.4.Бутстррап.....	55
6.5.Множественная проверка гипотез.....	55
6.5.1.Поправка Бонферони.....	56
6.5.2.Метод Холма.....	57
6.5.3.Метод Бенджамина-Хохберга (FDR).....	58
7. Регрессия.....	58
7.1.Линейная регрессия.....	58
7.2.Метрики качества в задачах регрессии.....	60
7.3.Переобучение и недообучение.....	62
7.4.Мультиколлинеарность признаков и регуляризация.....	62
7.5.Поиск гиперпараметров и кросс-валидация.....	65
7.6.Нелинейная регрессия.....	67
7.7.Линейная классификация.Логистическая регрессия.....	67
7.7.1.Логистическая регрессия.....	67
7.7.2.Метод опорных векторов (SVM).....	69
7.7.3.Метрики качества в задачах классификации.....	70
8.Анализ и прогноз квазипериодических рядов.....	74
8.1.Фильтрация.....	76
8.1.1.Фильтр с конечным импульсным откликом (КИХ-фильтр).....	76
8.1.2.Фильтр с бесконечным импульсным откликом (БИХ-фильтр).....	77
8.2.Авторегрессия.....	78
8.3.Стационарность временных рядов.....	80
8.4.Время корреляции временных рядов.....	85
8.5.Модели ARIMA и SARIMAX.....	87
8.6.Разложение по ортогональным системам функций.Спектры.....	91
8.6.1.Преобразование Хаара.....	92
8.6.2.Преобразование Уолша.....	94
8.6.3.Преобразование Фурье.....	96

8.7.Спектральная оценка периодических функций.....	98
8.7.1.Окноное преобразование Фурье.....	99
8.7.2.Параметрический спектр на основе ARMA.....	101
8.7.3.Метод MUSIC.....	102
8.8. Анализ нестационарных сигналов.....	104
8.8.1.Спектрограмма.....	104
8.8.2.Вейвлет-преобразование.....	105
9. Извлечение признаков.....	108
9.1.Методы без генерации новых признаков.....	108
9.1.1.Одномерные методы.....	108
9.1.2.Многомерные (жадные) методы.....	109
9.1.3.Методы на основе моделей.....	110
9.1.3.1.Комбинированное жадное добавление/удаление.....	110
9.1.3.2.Модель линейной регрессии.....	111
9.1.3.3.Модель решающих деревьев.....	112
9.1.3.4.Перестановочный метод.....	112
9.2.Методы с генерацией новых признаков.....	113
9.2.1.Линейные методы.....	114
9.2.1.1.Метод случайных проекций.....	114
9.2.1.2.Метод главных компонент.....	115
9.2.1.3.Неотрицательное матричное разложение.....	116
9.2.2.Нелинейные методы.....	119
9.2.2.1.Многомерное шкалирование(MDS).....	119
9.2.2.2.Стохастическая кодировка соседей(SNE).....	120
9.2.2.3.Стохастическая кодировка соседей по t-распределению(t-SNE).....	123
9.2.2.4.Аппроксимация и проекция равномерных многообразий (UMAP).....	124
9.2.3.Полиномиальное спрямляющее пространство.....	125
9.2.4.Автоэнкодеры.....	126
10.Аномалии.....	129
10.1.Поиск аномалий.....	129
10.1.1. Глобальные аномалии.....	129
10.1.1.1.Статистические тесты.....	129
10.1.1.2.Метод интервалов. Ящик с усами.....	130
10.1.1.3.Изолирующий лес.....	131
10.1.1.4.Одноклассовый метод опорных векторов.....	132
10.1.1.5.Эллиптические методы.....	133
10.1.1.6.Локальный уровень выброса, метрические кластеризаторы.....	134
10.1.2. Контекстные аномалии.....	135
10.1.3. Коллективные аномалии.....	136
10.2.Построение моделей, устойчивых к аномалиям в обучающих данных.....	138
11.Анализ оттока.....	139
11.1.Задача оттока с категориальными переменными.....	139
11.2.Задача оттока с числовыми переменными.....	139
12.Тематическое моделирование.....	140
12.1.Латентно-семантический анализ (LSA).....	141
12.2.Вероятностный латентно-семантический анализ (PLSA).....	142
12.3.Латентное размещение Дирихле (LDA).....	145
12.4.LDA с регуляризациями.BigARTM.....	147
Заключение.....	149
Литература.....	150

# **1. Введение**

Анализ данных — основа современных методов анализа и прогноза большинства современных направлений бизнеса, науки и многих других областей человеческой деятельности. Основу современного анализа данных составляют разнообразные математические методы, позволяющие выявлять, анализировать и доказывать эффекты и закономерности в данных, представленных в различном цифровом виде: числовом, аудио и видео формах. Эта книга сделана по материалам лекций, читавшихся автором в ИГУ, в ИТ Академии En+, в Малой школе анализа данных, в ИСЗФ СО РАН. Книга содержит программы на языке Питон, работоспособные на момент издания. Рекомендуется читать книгу одновременно с «Введение в Большие данные и машинное обучение (конспект лекций)» автора.

## **1.1. Виды анализа данных**

К основным видам анализа данных относят описательный, диагностический, прогностический и рекомендательный анализ. Каждый последующий тип анализа сложнее предыдущего, однако выше его по эффективности получаемых выводов и моделей. В данном курсе основной упор делается на первые три вида.

Подробнее можно посмотреть в [1].

### **1.1.1. Описательный анализ**

Это самый простой и распространенный тип аналитики, который используют при анализе данных. Он объединяет и выделяет закономерности в текущих и исторических данных. Описательная аналитика используется для визуализации ключевых показателей, которая позволяют отслеживать их тенденции. Описательная аналитика помогает понять, что происходит с данными в конкретный момент. Обычно описательный анализ включает в себя агрегирование данных, их первичный анализ и визуализацию.

Агрегирование данных – это сбор, сортировка и форматирование данных для упрощения их дальнейшего анализа.

Первичный анализ данных обычно включает в себя определение типа данных, пределов их изменений, выявление ошибок в данных, выявление их качественного поведения во времени и пространстве, а также выявление качественных и количественных соотношений между различными частями данных.

Одним из способов представления результатов описательной аналитики данных является их визуализация.

Основными шагами описательного анализа являются:

1. Определение метрик моделей данных. Задача этого шага - определить ключевые параметры (метрики), которые вы хотите изучать и которыми вы будете оценивать качество ваших данных и качество моделей их описывающих. Обычно метрики разделяются на бизнес-метрики и прокси-метрики. Бизнес-метрики — это клиент-ориентированные показатели, с помощью которых клиенты измеряют эффективность модели их процесса для своих задач. В приложении к бизнесу это чаще всего показатели издержки и прибыли. Прокси-метрики (далее в курсе - «метрики») — это математические показатели, позволяющие субъективно оценить улучшение или ухудшение бизнес-метрик. В машинном обучении чаще всего анализируются и моделируются прокси-метрики, выбираемые из соображений скорости их реакции на различные изменения в данных и простоты их расчетов. Задача перевода бизнес-

метрик в прокси-метрики выходит за рамки этого курса — предполагается что вам, как аналитикам данных, уже так или иначе известны прокси-метрики для вашей задачи, характерные для того вида деятельности, который вы анализируете.

2. Определение и поиск необходимых данных. Этот шаг связан с поиском и извлечением данных, необходимых для получения (расчета) желаемых параметров (метрик). Этот этап включает в себя поиск и анализ источников данных. Этот этап выходит за рамки курса и зависит от вида вашей будущей деятельности.
3. Извлечение и подготовка данных(переменных). Если данные поступают из нескольких источников, извлечение, объединение и подготовка данных для анализа является трудоемким процессом (по статистике он занимает до 95% всего времени, потраченного на анализ данных). Этот шаг может включать в себя очистку данных для устранения несоответствий и ошибок в данных из разных источников, а также преобразование данных в формат, подходящий для инструментов анализа. Этот этап частично выходит за рамки курса и зависит от вида представления ваших данных. В курсе рассматриваются базовые способы его применения.
4. Анализ данных. Описательная аналитика данных часто включает в себя применение основных математических операций или преобразований к одной или нескольким переменным, а также расчеты по ним необходимых метрик.
5. Представление данных. Это представление данных в понятных клиенту аудио-визуальных формах, используется для упрощения понимания особенностей изучаемых данных заинтересованными сторонами.

### **1.1.1.2. Типизация данных**

Критически важным при анализе данных является понимание типов данных с которыми вы работаете. В зависимости от типа данных различается набор методов, который может к ним применяться, а также могут меняться особенности обработки и моделирования данных. Использование методов и моделей, неадекватных реальным типам данных может приводить к неверным результатам, моделям и интерпретациям.

Существует несколько базовых классификаций данных.

**Классификация данных по способу их получения.** Данные делятся на сырье и обработанные. Обработка сырых данных более затратна, обработанные данные могут содержать ошибки различных видов, внесенные предыдущей обработкой. По возможности, рекомендуется при анализе использовать сырые данные.

**Классификация данных по мере их упорядоченности.** В основном данные делятся на упорядоченные, неупорядоченные и частично упорядоченные данные. Для упорядоченных данных важен порядок их следования (например по времени для аудиосигналов), для неупорядоченных данных порядок их следования неважен. Чаще всего однако исследователю приходится работать с частично-упорядоченными данными — вдоль каких-то параметров упорядоченность важна, вдоль других нет. Примером частично-упорядоченных данных может быть набор фотографий (порядок следования пикселов по горизонтали и вертикали важен, а порядок просмотра фотографий — нет).

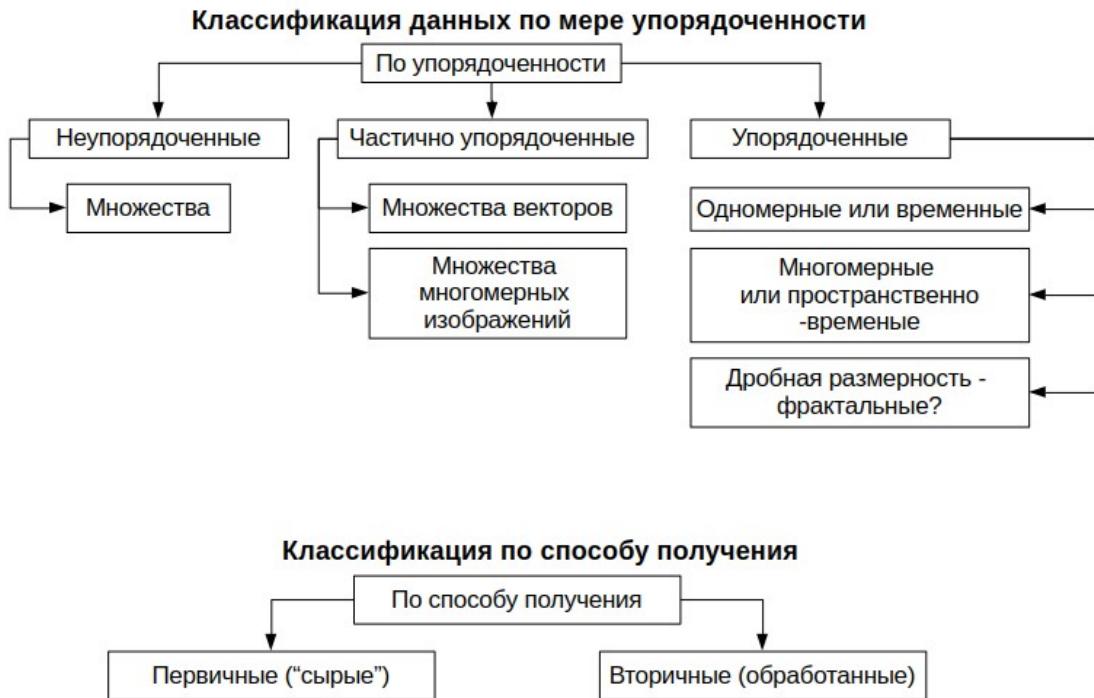


Рис.1. Типы классификации данных по мере упорядоченности и способу получения

Следующим важным способом классификации является **классификация данных по типу шкалы**. Здесь различают четыре основных типа — номинальные, порядковые, интервальные и шкалу отношений. Первые два типа являются категориальными типами, вторые два — количественными (или числовыми).



Рис.2. Классификация данных по типу шкалы

Номинальная шкала (категориальная, наименований) — это шкала измерения, которая используется для идентификации. Она является самой «слабой» из четырех видов шкал в смысле возможности обработки данных. Может использоваться только как метка. Единственный вид статистического анализа, который можно выполнить с использованием номинальной шкалы, это вычисление процентных долей, частот и моды. Данные в номинальной шкале можно проанализировать графически с помощью гистограммы и круговой диаграммы.

Порядковая шкала (ординальная, ранговая) — предполагает ранжирование (упорядочивание) значений переменной в зависимости от масштабирования. Атрибуты в порядковой шкале обычно располагаются в порядке возрастания или убывания некого качества. Она использует номинальную шкалу и порядковую операцию над ней «лучше чем»

или «больше чем». В добавление к методам номинальной шкалы в порядковой шкале можно использовать для статистического анализа квантильные статистики. Объекты можно сортировать и сравнивать, а также обрабатывать методиками основанными на ранге данных.

Интервальная шкала (разностей) — это шкала, в которой уровни упорядочены, а интервалы между ними не равны между собой. Интервальная шкала не только позволяет однозначно определить, какое значение больше (меньше), но и на сколько. В добавление к методам порядковой и номинальной шкал, в интервальной шкале могут выполняться арифметические операции над интервалами. Интервальную шкалу можно использовать при расчете среднего значения, медианы, моды, стандартного отклонения и других статистик. Объекты часто можно сравнивать, сортировать, складывать и умножать.

Шкала отношений (абсолютная) является «наивысшим» уровнем представления данных. Она может рассматриваться как расширение интервальной шкалы — либо использование равных интервалов, либо использование непрерывных величин. Шкала отношений совместима практически со всеми методами математического и статистического анализа.

### 1.1.1.2. Базовые методы агрегации данных

Источники данных могут быть очень разнообразны, и приведены на Рис.3. Источники данных важны в свете способов извлечения из них информации — протоколов и программных интерфейсов (API).



Рис.3. Источники данных

Важным при создании набора данных (датасета) является агрегация данных — объединение данных, полученных из разных источников. Программно проблема состоит в том, чтобы объединить разнородные данные и унифицировать к ним доступ. Основные программные подходы к агрегации приведены на Рис.4.

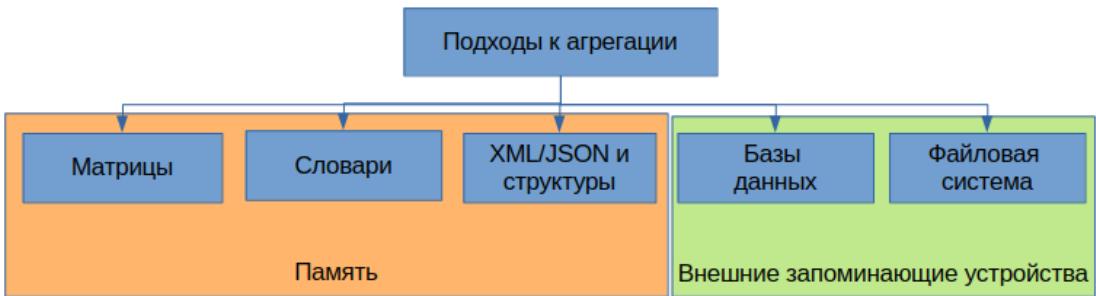


Рис.4.Программные подходы к агрегации данных

Важным при создании датасетов является приведение данных к единому масштабу (временному, пространственному и т.д.). Получающиеся при этом однородные данные

проще анализировать различными методами. Различают три основных подхода к решению этой задачи:

-Прореживание, когда все данные приводятся к масштабу самого редкого типа данных. Недостатком метода является выбрасывание значащих данных из типов, полученных с высоким разрешением;

-Интерполяция, когда все данные приводятся к масштабу самого часто полученного типа данных, а в остальные данные добавляются фиктивные данные полученные из модельных соображений (например гладкости изменений). Недостатком метода является его зависимость от валидности модели.

-Аппроксимация, когда все данные приводятся к масштабу самого часто полученного типа данных, и заменяются моделями, построенными с учетом измеренных данных. Недостатком метода является его еще большая зависимость от валидности модели, и отсутствие реально измеренных данных в аппроксимациях. Однако метод работоспособен при наличии хороших, адекватных данным, моделей.

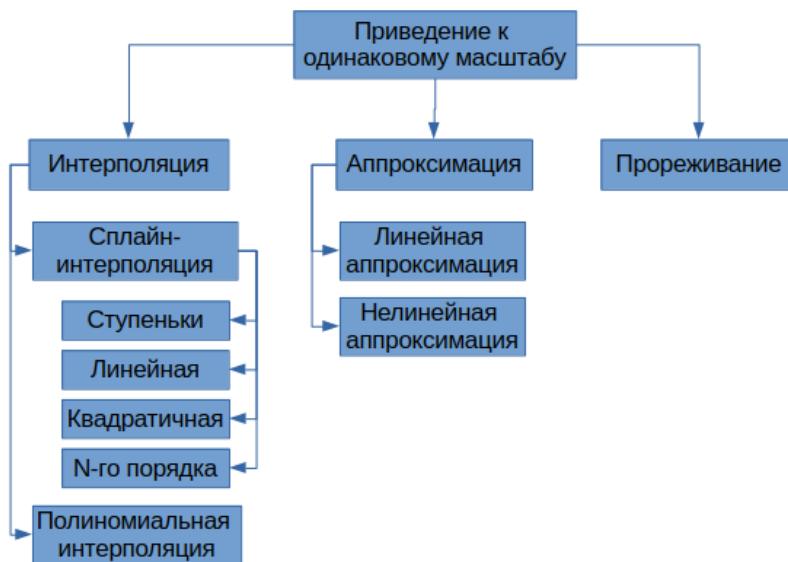


Рис.5. Методы приведения к единому масштабу

### 1.1.1.3.Методы визуализации

Методы визуализации делят на общую, специальную, концептуальную, стратегическую, метафорическую, и комбинированную визуализации.

Общая визуализация используется для представления количественной информации в традиционной схематической форме. Это круговые и линейные диаграммы, гистограммы и спектрограммы, таблицы и различные точечные и линейные графики.

Специальная визуализация - это специфические формы представления информации: карты и полярные графики, графики с параллельными осями, диаграммы Эйлера и др.

Концептуальная визуализация позволяет разрабатывать сложные концепции, отношения между объектами, идеи и планы с помощью концептуальных карт и других подобных видов диаграмм.

Стратегическая визуализация переводит в визуальную форму различные данные об аспектах работы организаций и бизнес-процессах. Это всевозможные диаграммы производительности, жизненного цикла и графики структур организаций.

Метафорическая визуализация используется для представления информации в виде

геометрических фигур и их композиций (например, значения признака представляются кругами разного размера) для улучшения иллюстративности выделяемого эффекта клиенту.

Комбинированная визуализация используется для объединения нескольких типов представлений данных в одну схему.

При анализе данных чаще всего используется общая визуализация.

### **1.1.2.Диагностический анализ**

Диагностический анализ (анализ причин) обеспечивает более глубокий анализ взаимоотношений между данными и их динамики, чтобы ответить на вопрос: почему это произошло?

Он включает использование таких подходов, как статистический и корреляционный анализ, спектральный анализ, регрессионный анализ. При диагностическом анализе выявляются зависимости между различными данными, что и помогает найти ключевые причины наблюдаемых явлений, и ключевые параметры, ответственные за наблюдаемые изменения. При этом проводится проверка различных гипотез и отбор наиболее правдоподобных из них.

### **1.1.3.Прогностический анализ**

Прогностический анализ это создание эффективной модели процессов, учитывающей реальные данные. Так называемое решение прямой задачи – прогноз ключевой величины по историческим и/или получаемым в текущем времени экспериментальным данным. Используется, чтобы предсказать, что произойдет дальше по известным значениям параметров.

Обычно для этого используются теоретические, эмпирические, математические или обучающиеся модели, наиболее адекватно описывающие существующие данные. В последнее время широко применяются модели машинного обучения типа нейронных сетей.

### **1.1.4.Рекомендательный анализ**

Рекомендательный анализ позволяет дать рекомендации, что можно изменить в текущих процессах, чтобы получить необходимое вам значение ключевого параметра в будущем. Он предлагает различные варианты действий и описывает возможные последствия для каждого из них.

В математике часто называется решением обратной задачи – по ожидаемому значению ключевого параметра определить, какими должны быть исходные значения параметров.

## **2.Вероятность**

### **2.1. Определение**

У вероятности есть несколько определений: одно классическое, другое аксиоматическое.

Классическое определение вероятности это отношение количества требуемых нам исходов эксперимента к общему количеству исходов при условии бесконечного числа проверок (экспериментов).

Аксиоматическое определение (колмогоровская вероятность) говорит что вероятность – это любая мера какого-то множества исходов (событий) которая удовлетворяет следующим условиям:

- она неотрицательна;
- она аддитивна по взаимоисключающим событиям;
- в сумме по всем возможным событиям она равна 1.

Это значит что какой бы набор событий вы не взяли, можно каждому поставить в соответствие какое-то число, которое больше либо равно нулю и меньше единицы, и его можно назвать вероятностью, если выполнены условия выше.

Если у вас есть два взаимоисключающих события (тех которые никогда не происходят одновременно), то вероятность того, что произойдет одно или другое равна сумме их вероятностей. Вероятность того, что произойдет любое из всех возможных взаимоисключающих событий равна 1. Такое определение называется аксиоматикой Колмогорова в теории вероятностей.

Необходимо понимать, что в выборе аксиоматической вероятности всегда есть произвол – вы можете поставить в соответствие событию практически любое числовую вероятность, а определение классической вероятности события требует очень большого (желательно бесконечного) числа наблюдений.

Поэтому аксиоматическая и классическая вероятности чаще всего будут отличаться друг от друга: при аксиоматической вероятности вы назначаете каждому событию его вероятность так чтобы итоговый результат соответствовал всем требованиям к вероятностному описанию, а при классическом – вы определяете вероятность экспериментально (или исходя из какой-то экспериментальной модели). Для объяснения многих понятий и на этапе анализа данных чаще всего используется классическое определение вероятности, а в машинном обучении чаще всего используется аксиоматическое определение вероятности.

### **2.2. Связанные события. Полная и условная вероятность.**

В задачах курса часто используется понятие полной и условной вероятности. Условная вероятность вводится в основном для связанных событий, когда вы хотите посмотреть как они между собой связаны и посчитать вероятность того, что величина  $x$  принимает какое-то значение при условии, что вы знаете, какое значение имеет величина  $y$ . Такая вероятность обозначается как  $P(x|y)$  и читается как вероятность наступления события  $x$  при условии что событие  $y$  уже произошло.

Полную вероятность события можно записать через условные:

$$P(y) = P(y|x)P(x) + P(y|\neg x)P(\neg x)$$

Можно доказать, что если у вас есть какие-либо события  $x, y$ , то выполняются:

$$P(x|y)P(y) = P(y|x)P(x)$$

Часто это выражение записывают в другом виде:

$$P(x|y) = \frac{P(y|x)P(x)}{P(y)}$$

Это выражение называется формулой Байеса и позволяет вам связать условные вероятности по разным переменным и их полные вероятности. Формула Байеса дает простейший способ оценивать разные типы вероятностей по другим, связанным с ними вероятностям.

### 3.Статистические распределения

Важным при изучении случайных величин является определение закономерностей их возникновения. Эти закономерности описываются многими характеристиками, наиболее важная из которых — статистические распределения, описывающие вероятность появления различных значений этой случайной величины в независимых экспериментах.

Распределения наиболее часто встречающихся случайных величин можно разделить на дискретные и непрерывные распределения. Дискретные распределения обычно характерны для категориальных и интервальных величин, непрерывные — для шкалы отношений (действительных чисел).

Рассмотрим базовые одномерные распределения, генерирующие за один раз одно случайное число.

#### 3.1. Дискретные распределения.

##### 3.1.1.Распределение Бернулли.

Распределение Бернулли можно рассматривать как бросание монеты, при том, что частота выпадения орла или решки могут различаться. Очевидно, что выпадения орла и решки — взаимоисключающие события и составляют полный набор возможных исходов. А значит полная вероятность, равная сумме вероятностей орла и решки, равна 1. Распределение может применяться для симуляции поведения объектов, имеющих только два значения — номинальных, порядковых и интервальных.

Таким образом, распределение Бернулли однопараметрическое, и этим единственным параметром является вероятность выпадения орла.

Вероятность появления  $P(x)$  значения  $x$  определяется по формуле:

$$P(x) = \begin{cases} 1-p, & \text{if } x=0 \\ p, & \text{if } x=1 \end{cases}$$

Сгенерируем случайные числа, имеющие распределение Бернулли. Будем использовать встроенный генератор случайных чисел, генерирующего случайные числа в диапазоне 0..1:

###### Код 1. Распределение Бернулли

```
import numpy as np
import matplotlib.pyplot as plt
def bernoulli(p=0.5):
    u = np.random.rand()
    return 1 if 0 <= u < p else 0
x=[bernoulli() for i in range(1000)]
plt.hist(x,bins=20)
plt.xlabel('value')
plt.ylabel('# of cases');
```

Необходимо заметить, что при работе с дискретными распределениями необходимо корректно выбрать число бинов в гистограмме, проще всего выбрать его в несколько раз больше числа уникальных значений переменной. Это позволит продемонстрировать дискретность распределения.

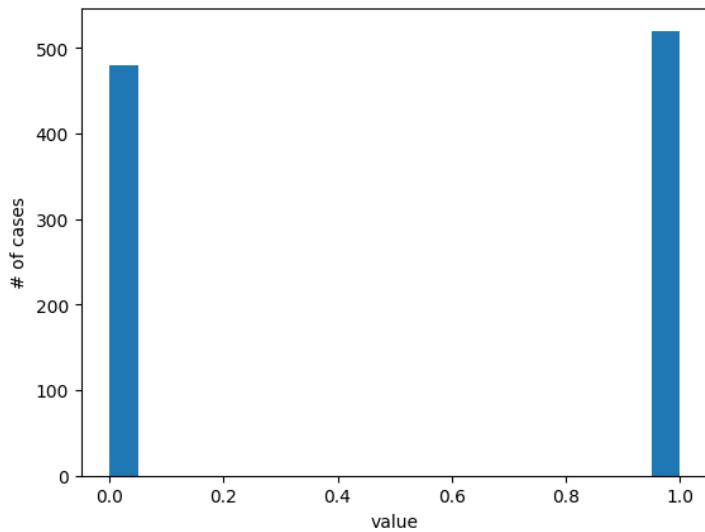


Рис.6. Появление случайных чисел 0 и 1, имеющих распределение Бернулли с параметром  $p=0.5$

### 3.1.2.Биномиальное распределение

Биномиальное распределение возникает в естественных случаях, если мы продолжаем монетку кидать много раз, но партиями равной длины, и считать количество выпадений орла. Например, если мы бросаем монетку по 10 раз, то варианты исходов случайны и могут быть от 0 до 10. Распределение такого числа исходов, если мы будем повторять наши эксперименты достаточно долго, и будет биномиальным распределением. Биномиальное распределение двухпараметрично, и определяется вероятностью выпадения орла  $p$  и числом бросков  $n$ . Распределение может применяться для симуляции поведения объектов, имеющих только п дискретных значений — номинальных, порядковых, и интервальных.

Вероятность появления  $P(x)$  значения  $x$  определяется по формуле:

$$P(x) = \frac{n!}{x!(n-x)!} p^x (1-p)^{n-x}$$

Построим биномиальное распределение, просуммировав  $n=6$  случайных чисел из распределения Бернулли с вероятностью  $p=0.5$ :

Код 2. Биномиальное распределение

```
def binom(n=3, p=0.5):
    u = np.random.rand(n)
    return np.sum(u < p)
x=[binom(6,0.5) for i in range(1000)]
plt.hist(x,bins=30)
```

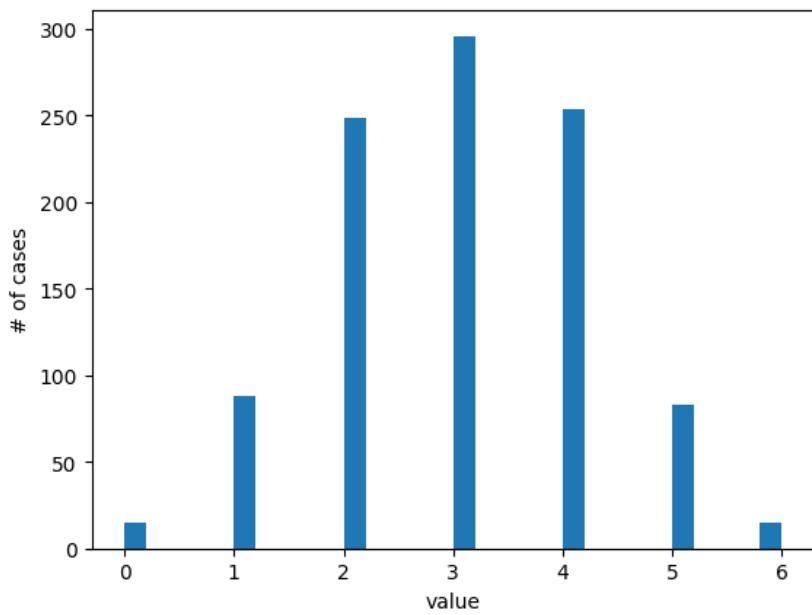


Рис.7. Появление случайных чисел, имеющих биномиальное распределение с параметрами  $n=6, p=0.5$

### 3.1.3. Распределение Пуассона

Распределение Пуассона может возникать, как число посетителей, проходящих через кассу за единицу времени, при условии что средний интервал  $T$  между ними известен (и интервалы распределены с экспоненциальной функцией распределения). Распределение Пуассона однопараметрично, и определяется средним числом посетителей за единицу времени  $\lambda$ .

Распределение может применяться для симуляции поведения объектов, имеющих счетное (бесконечное) число дискретных значений - номинальных, порядковых, и интервальных.

Вероятность появления значения  $x$  определяется по формуле:

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

Построим его:

#### Код 3. Распределение Пуассона

```
def poisson(lam=1):
    time,count = 0,0
    while time <= 1.0:
        u = np.random.rand()
        interval = -np.log(1 - u) / lam if u < 1.0 else float('inf')
        time += interval
        if time <= 1.0:
            count += 1
    return count
x=[poisson(3) for i in range(1000)]
plt.hist(x,bins=100)
```

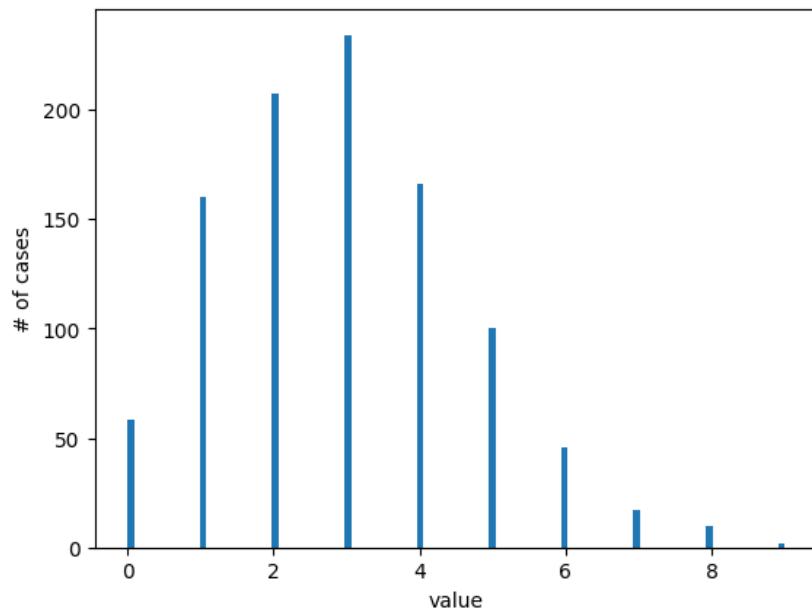


Рис.8. Появление случайных чисел, имеющих распределение Пуассона с параметром  $\lambda=3$

## 3.2.Непрерывные распределения.

### 3.2.1.Равномерное распределение.

Непрерывные распределения описывают события, которые могут принимать любое из непрерывного ряда значений. Одним из базовых является равномерное распределение.

Большинство простых программных генераторов случайных чисел имеет равномерное распределение. Мы его можем сгенерировать используя функцию `numpy.random.rand`.

Необходимо заметить, что при работе с непрерывными распределениями необходимо корректно выбрать число бинов в гистограмме, проще всего выбрать его равным корню из числа измерений(сэмплов). Параметрами равномерного распределения является его минимальное (a) и максимальное (b) значения.

Распределение может применяться для симуляции поведения объектов, имеющих непрерывные значения — действительных чисел из шкалы отношений.

#### Код 4. Равномерное распределение

```
def uniform(a=0,b=1):
    u = np.random.rand()
    return a+u*(b-a)
x=[uniform(2,4) for i in range(1000000)]
plt.hist(x,bins=1000)
```

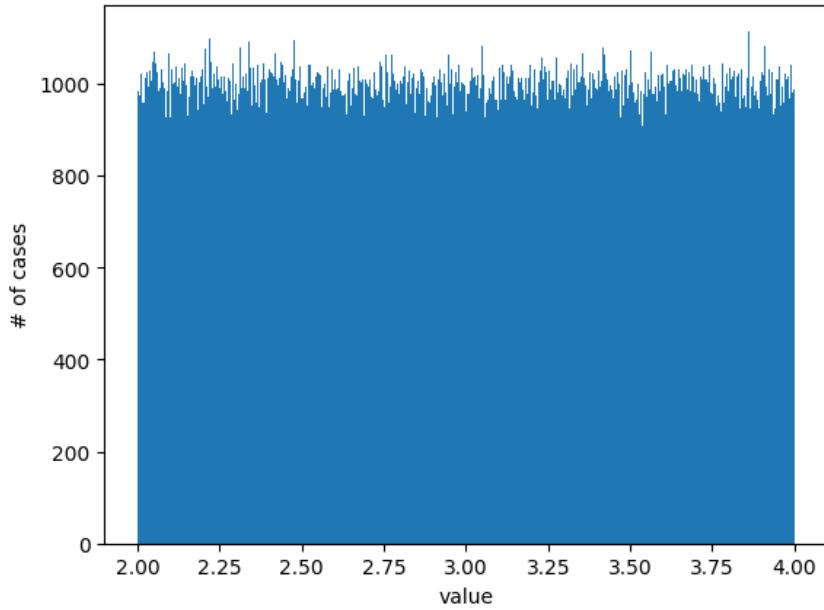


Рис.9. Появление случайных чисел, имеющих равномерное распределение с параметрами  $a=2, b=4$

Одна из основных проблем непрерывного распределения – невозможность посчитать вероятность какого-то события – она всегда будет равна нулю. Поэтому для описания непрерывных величин используют другие – непрерывные функции: функцию распределения и функцию плотности вероятности.

Функция распределения – это функция, равная вероятности того, что случайная величина не превышает значения ее аргумента:

$$F(y) = P(x \leq y)$$

Отсюда следуют основные свойства такой функции:

$$F(-\infty) = 0$$

$$F(\infty) = 1$$

$$F(x_1) \geq F(x_2), \text{ if } x_1 > x_2$$

Функция плотности вероятности  $p(x)$  определяет вероятность того, что случайная величина попадает в бесконечно малый промежуток значений, деленную на величину этого промежутка и имеет смысл вероятности, но для непрерывных величин. Функция распределения  $F(x)$  и плотность вероятности  $p(x)$  связаны интегрально-дифференциальными соотношениями:

$$F(x) = \int_{-\infty}^x p(y) dy$$

$$p(x) = \frac{dF(y)}{dy} \Big|_{y=x}$$

Для построения функции распределения и плотности вероятности необходимо нормировать интеграл на 1, что соответствует нормировкам:

$$\int_{-\infty}^{\infty} p(y) dy = 1$$

$$F(\infty) = 1$$

Для функции `hist` из библиотеки `matplotlib` это реализуется установкой параметра нормировки `density`

Код 5. Построение плотности вероятности

```
plt.hist(x,bins=1000,density=True);
```

Для равномерного распределения плотность вероятности имеет вид:

$$p(x) = \begin{cases} \frac{1}{b-a}, & \text{if } x \in [a, b] \\ 0, & \text{if } x \notin [a, b] \end{cases}$$

Функцию распределения можно построить, установив параметры накопления cumulative=True и нормировки density=True:

Код 6. Построение функции распределения

```
plt.hist(x,bins=1000,cumulative=True,density=True);
```

### 3.2.2.Нормальное распределение

Нормальное распределение — наиболее часто встречающееся распределение в обработке экспериментальных данных. Для нормального распределения плотность вероятности имеет вид:

$$p(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

У него два параметра — среднее  $\mu$  и среднеквадратичное отклонение  $\sigma$ .

Нормальное распределение может быть получено по аналогии с биномиальным — просуммируем достаточно большое количество случайных величин распределенных равномерно:

Код 7. Построение плотности вероятности нормального распределения со средним 1 и среднеквадратичным отклонением 2

```
def norm(mu=0, sigma=1, n_sum=12):
    u = np.random.rand(n_sum)-0.5
    std=u.std()*np.sqrt(n_sum)
    s=np.sum(u)
    res = (s/std*sigma+mu)
    return res
x=[norm(1,2,40) for i in range(10000)]
plt.hist(x,bins=100,density=True);
```

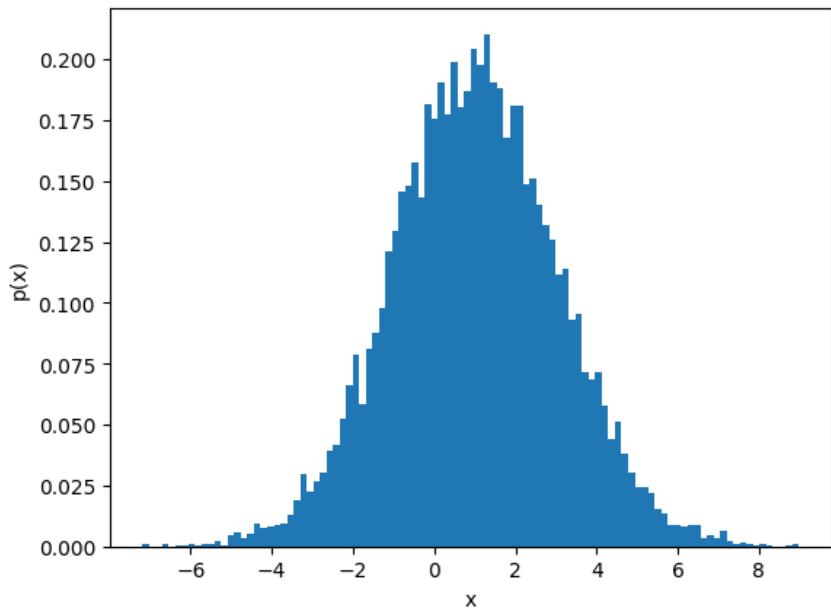


Рис.10. Плотность вероятности случайных чисел, имеющих нормальное распределение со средним 1 и среднеквадратичным отклонением 2

Чем большее число величин мы просуммировали, тем ближе получившееся распределение к нормальному. Это называется центральной предельной теоремой.

Среднее  $\bar{x}$  и дисперсия  $Dx$  непрерывной случайной величины рассчитывается по плотности ее вероятности по формулам:

$$\mu = \bar{x} = \langle x \rangle = \int_{-\infty}^{\infty} x p(x) dx$$

$$Dx = \langle (x - \langle x \rangle)^2 \rangle = \int_{-\infty}^{\infty} (x - \langle x \rangle)^2 p(x) dx$$

Среднеквадратичное отклонение – это квадратный корень из дисперсии и имеет смысл полуширины распределения.

$$\sigma = \sqrt{Dx}$$

По экспериментальным данным среднее и дисперсию можно рассчитать по формулам:

$$\langle x \rangle = \frac{1}{N} \sum_{i=1}^N x_i$$

$$Dx = \frac{1}{N} \sum_{i=1}^N (x_i - \langle x \rangle)^2 = \left( \frac{1}{N} \sum_{i=1}^N x_i^2 \right) - (\langle x \rangle)^2$$

Эта оценка дисперсии смещена, для несмещенной оценки нужно использовать:

$$\tilde{D}x = \frac{N}{N-1} Dx$$

### 3.2.3. Распределение хи-квадрат

Распределение Хи-квадрат часто используется при сравнении случайных чисел и их распределений. Хи-квадрат – это распределение суммы квадратов  $N$  нормальных величин с нулевым средним и единичной дисперсией. Количество суммируемых случайных величин называется числом степеней свободы.

Плотность вероятности такой случайной величины с  $k$  степенями свободы имеет вид:

$$p(x) = \frac{1}{2^{k/2} \Gamma(k/2)} x^{k/2-1} e^{-x/2}$$

Здесь  $\Gamma(k/2)$  - некая нормирующая функция (гамма-функция).

Сгенерируем и построим его используя генератор случайных чисел с нормальным распределением `numpy.random.randn`:

Код 8. Построение плотности вероятности распределения хи-квадрат с четырьмя степенями свободы

```
def chi2(df=1):
    u = np.random.randn(df)
    s=np.sum(u**2)
    return s
x=[chi2(4) for i in range(10000)]
plt.hist(x,bins=100,density=True);
```

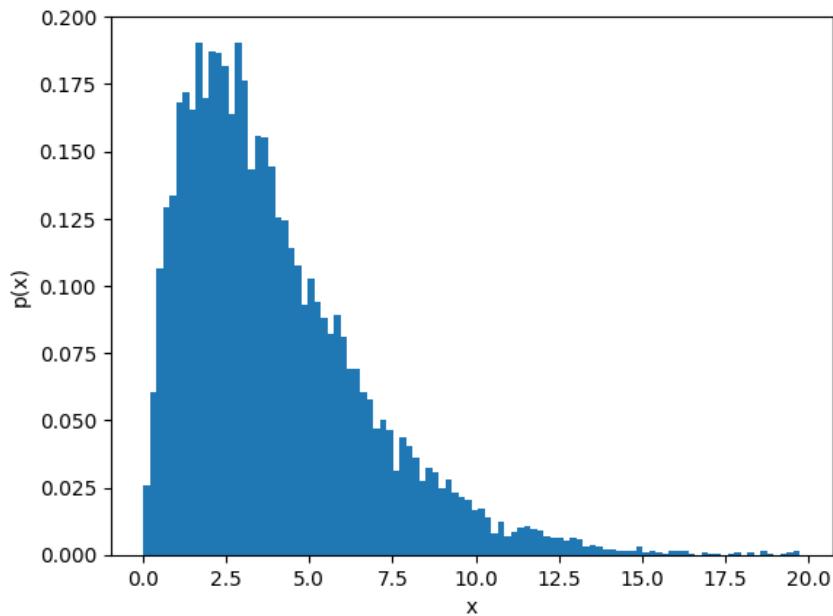


Рис.10. Плотность вероятности случайных чисел, имеющих распределение хи-квадрат с четырьмя степенями свободы

### 3.3. Центральная предельная теорема

Центральная предельная теорема утверждает следующее: если у вас случайная величина  $x$  распределена по какому-то закону распределения со средним  $\mu_x$  и среднеквадратичным отклонением  $\sigma_x$ , то если вы просуммируйте или усредните достаточно много таких случайных величин, их сумма будет иметь распределение близкое к нормальному, причем его параметры определяются по параметрам  $\mu_x, \sigma_x$  и числу величин, участвовавших в суммировании.

Центральная предельная теорема имеет две формулировки – для среднего и для

суммы. Если

$$x \sim P, E_x = \mu_x, D_x = \sigma_x^2$$

то

$$\sum_{i=1}^n x_i \sim N(n\mu_x, \sqrt{n}\sigma_x)$$

$$\frac{1}{n} \sum_{i=1}^n x_i \sim N(\mu_x, \frac{\sigma_x}{\sqrt{n}})$$

Знак  $\sim$  означает «иметь распределением» — величина слева от знака имеет распределение справа от знака.  $P$  - некое распределение,  $N(a,b)$  - нормальное распределение со средним  $a$  и среднеквадратичным отклонением  $b$ .

Можно проиллюстрировать работу центральной предельной теоремы кодом:

#### Код 9. Проверка центральной предельной теоремы

```
u = np.random.rand(10000,5)+10
mu_x, std_x = u.mean(), u.std()
n = u.sum(axis=-1)
n1 = np.random.randn(10000)*std_x*np.sqrt(5)+mu_x*5
plt.hist(n, bins=100, density=True, label='sum of x from U');
plt.hist(n1, bins=100, density=True, histtype='step', label='x from N');
plt.legend()
```

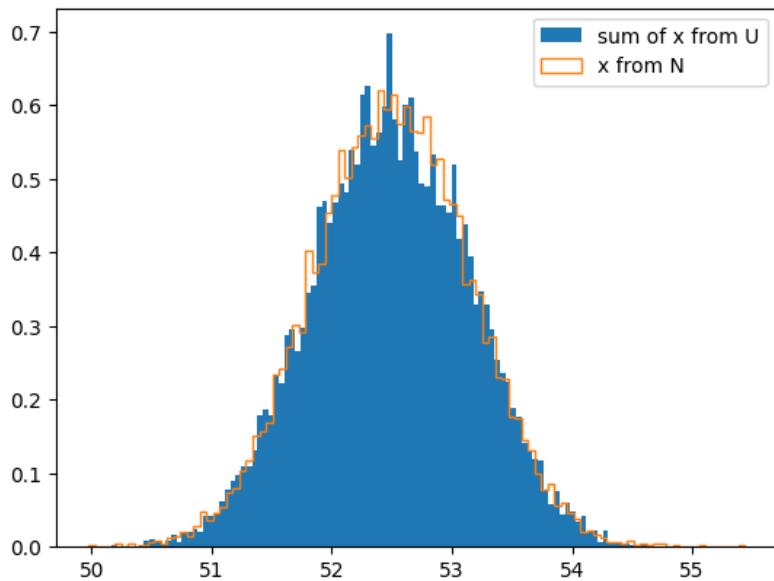


Рис.11.Проверка центральной предельной теоремы

## 3.4. Ядерное сглаживание

Часто данных недостаточно, чтобы построить плавную функцию плотности вероятности. В таких случаях используют ядерное сглаживание. В отличие от гистограммы, где интервал значений разбивается на непересекающиеся участки – бины, ядерное

сглаживание можно рассматривать как гистограмму, где ширина бина фиксирована, а его положение произвольно меняется. Можно обобщить этот подход на случай произвольного ядра  $K$ , которое взвешивает частоту появления чисел в интересующем нас диапазоне – бине:

$$p_k(y_i) = \frac{\sum_{j=1}^N K(x_j - y_i)}{N \int_{-\infty}^{\infty} K(x) dx}$$

Пример кода с прямоугольным окном

#### Код 10. Пример ядерного сглаживания с прямоугольным ядром

```
u = np.random.randn(1000)
plt.hist(u,bins=np.linspace(-5,5,200),density=True,label='original p(x)');
kx,D=[],1
for x in np.linspace(-5,5,100):
    kx.append([x,u[np.logical_and(x-D/2<u,u<x+D/2)].shape[0]])
kx=np.array(kx)
kx[:,1]=kx[:,1]/(D*1000)
plt.plot(kx[:,0],kx[:,1],label='kernel smooth');
plt.xlabel('x'),plt.ylabel('p(x)')
plt.legend()
```

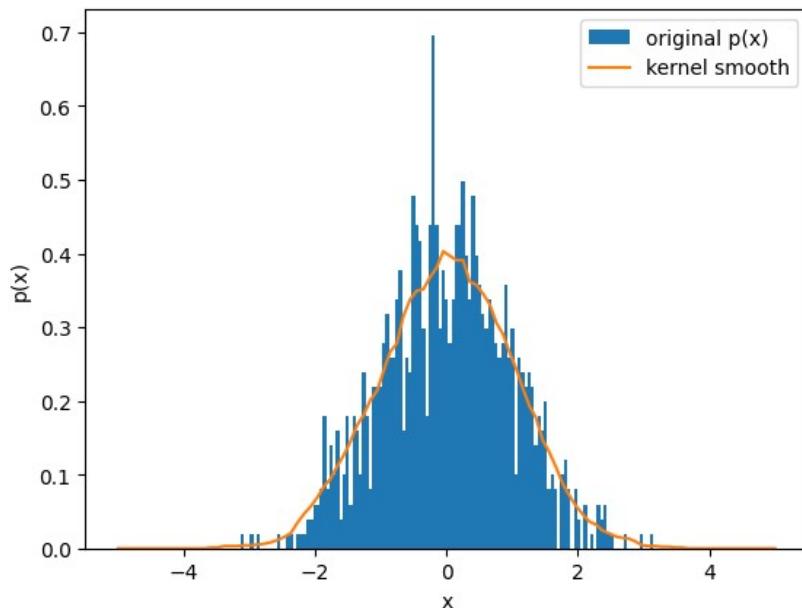


Рис.12. Пример ядерного сглаживания прямоугольным окном

## 3.5. Построение распределений в библиотеке scipy.

### 3.5.1. Статистики распределений

Рассмотрим основные функции `scipy.stats` для работы со случайными величинами. Библиотека `scipy.stats` обеспечивает единый интерфейс для работы со случайными

величинами различных распределений. Выбор вида распределения:

- `dist=scipy.stats.norm(0, 2)` – нормальное распределение с нулевым средним и СКО равным 2
- `dist=scipy.stats.uniform()` - равномерное распределение и многие другие

Функции для работы с распределениями:

- `dist.rvs (random value set)` – выбор заданного количества случайных величин, принадлежащих заданному распределению
- `dist.cdf (cumulative distribution function)` – построение функции распределения для заданного распределения в заданных точках
- `dist.pdf (probability distribution function)` - построение плотности вероятности для заданного распределения в заданных точках
- `dist.ppf (Percent point function)` – определение значения по значению функции распределения. Функция распределения монотонно неубывает, поэтому такое преобразование почти всегда однозначно.

Рассмотрим пример применения этих функций на примере распределения хи-квадрат с 4 степенями свободы:

#### Код 11. Пример применения pdf из библиотеки `scipy.stats`

```
import scipy.stats as ss
x=[chi2(4) for i in range(10000)]
plt.hist(x,bins=100,density=True,label='experiment')
dist=ss.chi2(df=4)
x2=np.linspace(0,20,100)
plt.plot(x2,dist.pdf(x2),label='scipy.stats.pdf');
```

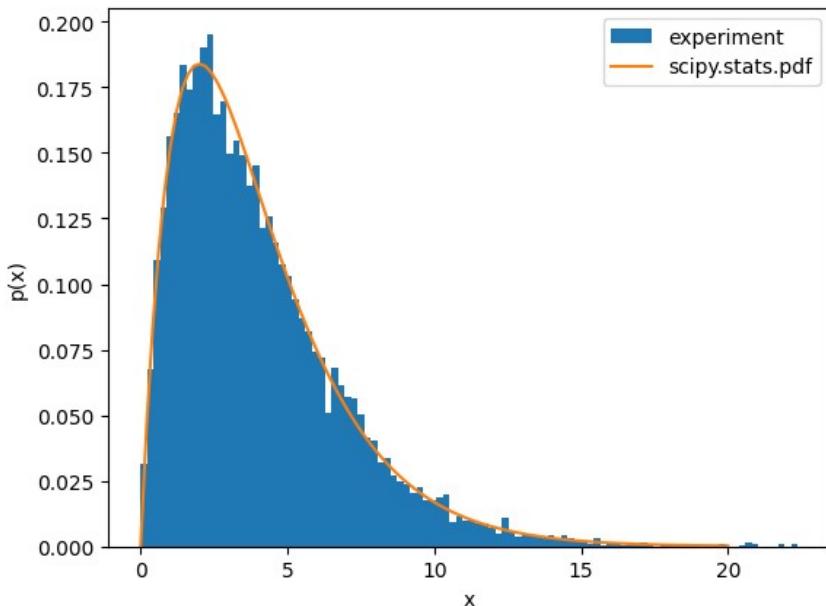


Рис.13.Пример применения pdf из библиотеки `scipy.stats`

### 3.5.2. Доверительные интервалы и квантили. Уровень значимости.

Квантиль – это интервал  $(-\infty, Q]$ , в который попадает заданная доля значений случайной величины по бесконечно большой выборке. Обычно квантиль обозначается как  $Q_\alpha$ , где  $\alpha$  – ожидаемая доля или уровень значимости. Таким образом:

$$\int_{-\infty}^{Q_\alpha} p(x) dx = \alpha$$

Есть три выделенных типа квантилей:

- Квартили –  $Q_{0.25}, Q_{0.5}, Q_{0.75}$ . Квартиль  $Q_{0.5}$  имеет собственное значение – медиана и смысл – половина значений в большой выборке больше этой величины, половина – меньше.
- Децили –  $Q_{0.1}, Q_{0.2}, \dots, Q_{0.8}, Q_{0.9}$ . В каждый дециль  $[Q_x, Q_{x+0.1}]$  попадает 10% всех значений в выборке.
- Перцентили –  $Q_{0.01}, Q_{0.02}, \dots, Q_{0.98}, Q_{0.99}$ . В каждый перцентиль  $[Q_x, Q_{x+0.01}]$  попадает 1% всех значений в выборке

Если распределение случайной величины задано, квантиля можно определять по .ppf если имеются только выборка значений – то используя nptrum.percentile():

Код 12. Сравнение перцентилей, полученное разными способами

```
x=np.random.randn(100000)*3+2
dist=ss.norm(2,3)
print('Q1:',np.percentile(x,25),dist.ppf(0.25))
#
Q1: -0.0339 -0.0234
```

Доверительным интервалом с уровнем доверия  $p$  является интервал, в который попадает доля  $p$  значений в большой выборке. Обычно различают три типа доверительных интервалов – правосторонний, левосторонний и двухсторонний.

Двухсторонний доверительный интервал уровня доверия  $p$  имеет вид:  $[Q_{(1-p)/2}, Q_{1-(1-p)/2}]$

Нижний (левосторонний) доверительный интервал уровня доверия  $p$  имеет вид:  $[Q_{1-p}, \infty)$

Верхний (правосторонний) доверительный интервал уровня доверия  $p$  имеет вид:  $(-\infty, Q_p]$

Двухсторонний доверительный интервал уровня доверия 0.5 называется интерквартильным расстоянием, в него попадает 50% всех значений, половина из которых меньше медианы, а половина – больше.

Построить доверительные интервалы по заданному распределению можно при помощи .ppf, а по заданной выборке – при помощи np.percentile.

Кроме доверительных интервалов в статистике используются среднее и среднеквадратичное отклонение.

Для нормального распределения в интервал  $[\langle x \rangle - \sigma, \langle x \rangle + \sigma]$  попадает примерно 68% всех значений в большой выборке, в интервал  $[\langle x \rangle - 2\sigma, \langle x \rangle + 2\sigma]$  – примерно 95% всех значений, а в интервал  $[\langle x \rangle - 3\sigma, \langle x \rangle + 3\sigma]$  – примерно 99.7% всех значений.

Для дискретных случайных величин можно использовать моду – наиболее вероятное значение.

## 4.Корреляционный анализ

### 4.1.Связь числовых величин

#### 4.1.1.Корреляция Пирсона

Если две случайных величины  $x, y$  связаны линейным соотношением:

$$y = A + Bx + \epsilon$$

где  $\epsilon$  достаточно мало, а  $A, B$  — некие постоянные, то можно говорить о линейной связи между этими случайными величинами.

Для выявления линейной связи используется величина

$$R = \frac{\sum_i (y_i - \langle y \rangle)(x_i - \langle x \rangle)}{\sigma_x \sigma_y}$$

Эта величина называется коэффициентом корреляции Пирсона. Она принимает значения  $[-1, 1]$ . При этом если  $|R|=1$ , то связь между двумя величинами не содержит малой величины  $\epsilon$ :

$$y = A + Bx,$$

при этом если  $B>0$  то  $R=1$ , а если  $B<0$  то  $R=-1$ .

Если  $R=0$ , то связь между двумя величинами отсутствует:

$$y = A + \epsilon$$

Рассмотрим вычисление коэффициента Пирсона:

#### Код 13. Расчет коэффициента корреляции Пирсона

```
def pearson(x, y):
    mx, my = np.mean(x), np.mean(y)
    sx, sy = np.std(x), np.std(y)
    cov = np.mean((x - mx) * (y - my))
    return cov / (sx * sy)
x = np.random.randn(100)
y = 5*x + 4 + np.random.randn(100)*0.1
print(pearson(x,y))
#
0.9998
```

Видно, что коэффициент Пирсона близок к единице, что говорит о линейной зависимости со слабой шумовой добавкой.

Таким образом, коэффициент корреляции Пирсона используется для обнаружения линейной взаимосвязи между двумя случайными величинами, чем больше по модулю коэффициент корреляции, тем меньше ошибка  $\epsilon$  в уравнении, связывающем их линейно.

Недостатки выборочного коэффициента Пирсона: для распределений, отличных от нормального, перестаёт быть эффективной оценкой популяционного коэффициента корреляции; служит мерой только линейной взаимосвязи; неустойчив к выбросам.

#### 4.1.2.Корреляция Спирмена

Проблему составляют случаи, когда связь между двумя величинами существует, но она нелинейна. Например  $y=x^2$ . Частным случаем такой связи является монотонная связь,

когда

$$y = f(x) + \epsilon : df/dx \geq 0$$

или

$$y = f(x) + \epsilon : df/dx \leq 0$$

В таком случае часто коэффициент Пирсона ошибается. Для таких случаев удобно использовать порядковую (ранговую) корреляцию. Рангом  $\text{rank}(x)$  случайной величины  $x$  в выборке называется номер ее позиции в упорядоченном ряду значений.

Коэффициент корреляции Спирмена – это коэффициент корреляции Пирсона для рангов случайных величин:

$$R_{\text{spearman}}(x, y) = R_{\text{pearson}}(\text{rank}(x), \text{rank}(y))$$

Код 14. Сравнение коэффициента корреляции Спирмена и Пирсона в случае нелинейной связи между случайными величинами

```
def rank(x):
    sx=np.sort(x)[::-1]
    r=[]
    for tx in list(x):
        r.append(sx[sx<=tx].shape[0])
    return np.array(r)
def spearman_correlation(x, y):
    rx = rank(x)
    ry = rank(y)
    return pearson(rx,ry)
y=np.exp(x)
print(pearson(x,y),spearman_correlation(x,y))
#
0.842 1.0
```

Видно, что в этом модельном случае коэффициент Пирсона меньше коэффициента Спирмена, что говорит о наличии монотонной, но не линейной, взаимосвязи между случайными величинами.

Коэффициент Спирмена выявляет монотонную взаимосвязь между случайными величинами  $y$  и  $x$ . Если связующая функция монотонно возрастает, то коэффициент Спирмена положителен, если монотонно убывает, то отрицателен. Значение  $+/- 1$  означает точную зависимость ( $\epsilon=0$ ), значение 0 означает отсутствие монотонной зависимости  $y=\epsilon$ .

## 4.2.Связь номинальных величин

При сравнении номинальных величин одним из основных объектов является матрица (таблица) сопряженности. Таблица сопряженности – это число наблюдений различных пар категорий:

	J1	J2	J3	
I1	50	10	20	$n_{0,s}=80$
I2	20	60	20	$n_{1,s}=100$

	$n_{m,0}=70$	$n_{m,1}=70$	$n_{m,2}=40$	$n=180$
--	--------------	--------------	--------------	---------

Рис.14. Пример таблицы сопряженности

Сложностью сравнения номинальных величин является требование инвариантности метрик к перестановкам порядка категорий.

Рассмотрим метрики, использующие анализ таблицы сопряженности.

#### 4.2.1.Коэффициент V Крамера.

Работает в предположении что значения распределены нормально.

$$\phi_c = \sqrt{\frac{\chi^2}{n(\min(K_1, K_2) - 1)}}$$

где  $\chi^2$  - значение критерия хи-квадрат по таблице сопряженности, проверяющее ее принадлежность к одному распределению.

Вычислим по матрице коэффициент V Крамера:

##### Код 15. Вычисление коэффициента V Крамера

```
from scipy.stats import chi2_contingency
def v_kramer_coefficient(cm):
    chi2, _, _, _ = chi2_contingency(cm)
    n = np.sum(cm)
    k = min(cm.shape)
    v = np.sqrt(chi2 / (n * (k - 1)))
    return v
cm= np.array([[50, 10,20],[20, 60,20]])
print (v_kramer_coefficient(cm))
#
0.51
```

Коэффициент V Крамера принимает значение 0.51, что говорит о невысокой корреляции между величинами.

#### 4.2.2.Индекс Рэнда

В отличие от коэффициента V Крамера, индекс Рэнда не требует нормальности распределений, он требует чтобы количество элементов в каждой категории было примерно одинаковое. Определяется, как количество пар значений, которые наблюдаются совместно:

$$RI = \frac{\sum_i a_i + \sum_j b_j}{C_2^n}$$

Здесь  $a$  — число пар, одновременно принадлежащих к одной категории для разных номинальных величин,  $b$  — число пар, одновременно принадлежащих разным категориям для разных номинальных величин.

#### Код 16. Вычисление индекса Рэнда

```
def rand_score_cm(cm):
    a = np.sum(cm * (cm - 1)) // 2
    n = np.sum(cm)
    sum_comb_rows=np.sum([np.sum(row)*(np.sum(row)-1)//2 for row in cm])
    sum_comb_cols=np.sum([np.sum(col)*(np.sum(col)-1)//2 for col in cm.T])
    total_pairs = n * (n - 1) // 2
    b = total_pairs - (sum_comb_rows + sum_comb_cols - a)
    return (a + b) / total_pairs
print(rand_score_cm(cm))
#
0.596
```

Индекс Рэнда достигает на этой матрице значения 0.60, что говорит о невысокой корреляции этих номинальных величин.

#### 4.2.3.Скорректированный индекс Рэнда

При дисбалансе категорий удобнее использовать скорректированный индекс Рэнда:

$$ARI = \frac{\sum_{i,j} C_2^{n_{ij}} - \left[ \sum_i C_2^{a_i} \sum_j C_2^{b_j} \right] / C_2^n}{\frac{1}{2} \left[ \sum_i C_2^{a_i} + \sum_j C_2^{b_j} \right] - \left[ \sum_i C_2^{a_i} \sum_j C_2^{b_j} \right] / C_2^n}$$

#### Код 17. Вычисление скорректированного индекса Рэнда

```
def comb2(a): return (a * (a - 1)).sum() / 2
def adjusted_rand_score_cm(confusion_matrix):
    cm = np.asarray(confusion_matrix)
    n = np.sum(cm) # Total number of samples
    sum_comb_rows = np.sum([comb2(np.sum(row)) for row in cm])
    sum_comb_cols = np.sum([comb2(np.sum(col)) for col in cm.T])
    index = np.sum([comb2(x) for x in cm.flatten()])
    expected_index = sum_comb_rows * sum_comb_cols / comb2(n)
    max_index = 0.5 * (sum_comb_rows + sum_comb_cols)
    if max_index == expected_index: ari = 0.0
    else:
        ari = (index - expected_index) / (max_index - expected_index)
    return float(ari)
adjusted_rand_score_cm(cm)
#
0.194
```

Скорректированный индекс Рэнда достигает на этой матрице значения 0.19, что подтверждает слабую зависимость переменных и является более адекватной оценкой при существенном дисбалансе классов (ведь мы по факту сравниваем 2 и 3 класса!). Для

полностью независимых величин индекс скорректированный Рэнда принимает оклонулевые значения.

### **4.3.Связь порядковых и числовых(или интервальных) величин**

Порядковые (категориальные) и интервальные (числовые) величины могут быть ранжированы по значениям, поэтому для выявления между ними взаимосвязи можно использовать коэффициент Спирмена.

### **4.4.Связь номинальных и числовых величин**

Является самой сложной задачей, поскольку одна из переменных не обладает порядком, а другая – обладает. Проще всего это проверить визуально, с помощью ящика с усами или с помощью метода интервалов – если ни один 95% доверительный интервал значений числовой величины в данной категории не пересекается с остальными доверительными интервалами – то присутствует существенная зависимость, обратное неверно и требует более сложных методов проверки методами проверки статистических гипотез или дискретизацией числовых величин, обсуждаемых далее при решении задачи оттока.

## **5.Кластеризация**

### **5.1.Введение**

Часто возникает задача разделить объекты на группы по степени похожести. Задача это может решаться двумя способами. Первый способ - с учителем, когда вам известно к какой группе отнести те или иные объекты и вам необходимо обучить некую решающую схему, которая для каждого нового объекта будет определять группу, к которой можно его отнести. Такая решающая схема будет называться классификатором, а группы – классами. Второй способ (без учителя) используется, когда неизвестно, какие объекты отнести к каким группам, и вам необходимо проанализировать ваши данные и выделить эти группы и объекты, к ним принадлежащие. Метод, который вы будете при этом использовать будет называться кластеризатором, а группы, которые вы выделили – кластерами.

Проще всего объекты можно считать точками в некотором пространстве их параметров. Существует много способов провести кластеризацию, в зависимости от того как вы определяете степень похожести точек и каким способом объединяете эти точки в группы.

Разделить методы можно на две категории: плоские алгоритмы и иерархические алгоритмы.

Необходимо заметить, что любой алгоритм кластеризации представляет собой какую-то модель данных и имеет по крайней мере один гиперпараметр, от которого зависит то, как данные будут разделены на кластера.

Существует также деление на четкие и нечеткие кластеризации, или жесткую и мягкую кластеризацию. Жесткая (четкая) кластеризация решает то, что объект однозначно принадлежит какому-то классу. Мягкая (нечеткая) кластеризация рассчитывает вероятности того, что объект принадлежит к тому или иному классу из всех доступных.

Подробнее можно посмотреть [2-3].

### **5.2.Плоские алгоритмы**

Плоские алгоритмы позволяют однозначно разделить набор точек на несколько кластеров при заданном значении гиперпараметра.

#### **5.2.1.Метрические методы кластеризации.**

Для метрических методов самое главное указать степень похожести двух точек, то есть ввести понятие непохожести или расстояния между точками. Чем меньше расстояние, тем более похожи точки друг на друга. Одним из наиболее простых и широко используемых метрических методов является метод DBSCAN.

##### **5.2.1.1.DBSCAN**

Рассмотрим самый простой случай, когда объекты представляют из себя точки на некоторой плоскости (или в евклидовом пространстве). Соответственно мы можем ввести расстояние, как степень похожести, обычно это евклидово расстояние, как корень из суммы квадратов разностей координат. Все метрические методы основаны на понятии расстояния. Когда мы ввели расстояние, можно кластеризовать наши данные с точки зрения этого расстояния – чем оно меньше, тем более похожи точки.

Гиперпараметром в методе DBSCAN выступает максимальное расстояние  $\epsilon$ , при котором две точки можно считать похожими (принадлежащими к одному кластеру). Метод

DBSCAN основан на простом правиле – если между двумя точками расстояние меньше чем  $\epsilon$ , то они принадлежат к одному кластеру. Обратное неверно.

Таким образом, если две точки можно соединить ломанной по другим точкам такой что все ребра этой ломанной меньше  $\epsilon$ , то они принадлежат к одному кластеру.

#### Код 18. Кластеризация искусственного датасета алгоритмом DBSCAN

```
from sklearn.datasets import make_blobs
from sklearn.cluster import DBSCAN
X, y = make_blobs(n_samples=500, centers=3, n_features=2, random_state=42)
dbSCAN = DBSCAN(eps=1)
clusters = dbSCAN.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=clusters, s=10)
```

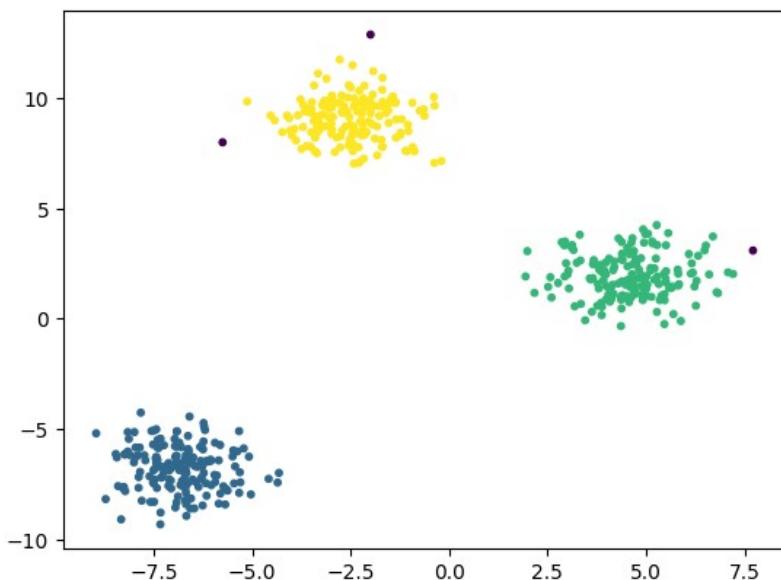


Рис.14. Кластеризация искусственного датасета алгоритмом DBSCAN

Существуют ситуации, когда вы не можете отобразить объекты в евклидовом пространстве, хотя можете указать попарные расстояния между ними (матрицу смежности). Например если требуется кластеризовать слова по метрике редакторского расстояния (расстояние Левенштайн). Большинство метрических методов кластеризации позволяют сделать это. Для этого необходимо подать на вход алгоритма не координаты каждой точки, а матрицу смежности – матрицу расстояний между каждой парой точек в датасете. В библиотеке sklearn для алгоритма DBSCAN это реализуется следующим образом:

#### Код 19. Кластеризация искусственного датасета алгоритмом DBSCAN на основе матрицы смежности

```
from sklearn.metrics.pairwise import euclidean_distances
from sklearn.cluster import dbscan
dist = euclidean_distances(X)
_,clusters = dbscan(dist,eps=1,metric='precomputed')
plt.scatter(X[:, 0], X[:, 1], c=clusters, s=10)
```

Легко убедиться, что результат кластеризации не изменился, хотя мы использовали не координаты, а только расстояния между парами точек.

Основным недостатком всех метрических алгоритмов является их неустойчивость к изменению масштабов осей, что связано с постоянством евклидового расстояния при перестановке координат.

Для повышения устойчивости перед кластеризацией рекомендуется нормировка каждой оси, чтобы величина проекции датасета на каждую ось была примерно одинаковой. Для этого можно использовать либо нормализацию (когда все координаты нормируются на диапазон [0,1]), либо стандартизацию (когда все координаты нормируются таким образом, чтобы по датасету среднее значение каждой координаты было равно нулю, а дисперсия – единице). Недостатком автоматической нормировки является то, что если данные будут представлены в одном масштабе по каждой оси и евклидово расстояние корректно для их описания, а наблюдаемая “вытянутость” есть внутреннее свойство датасета, то нормировкой можно исказить реальные кластера. Поэтому необходимость нормировки зависит от постановки задачи, и решение о ней должно приниматься обработчиком данных, исходя из условий задачи.

#### Код 20. Сравнение стандартного масштабирования и минимаксного масштабирования.

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler  
XS=StandardScaler().fit_transform(X)  
XN=MinMaxScaler().fit_transform(X)  
plt.scatter(XS[:, 0], XS[:, 1], s=10,label='standard scaler')  
plt.scatter(XN[:, 0], XN[:, 1], s=10,label='min/max scaler')  
plt.legend()
```

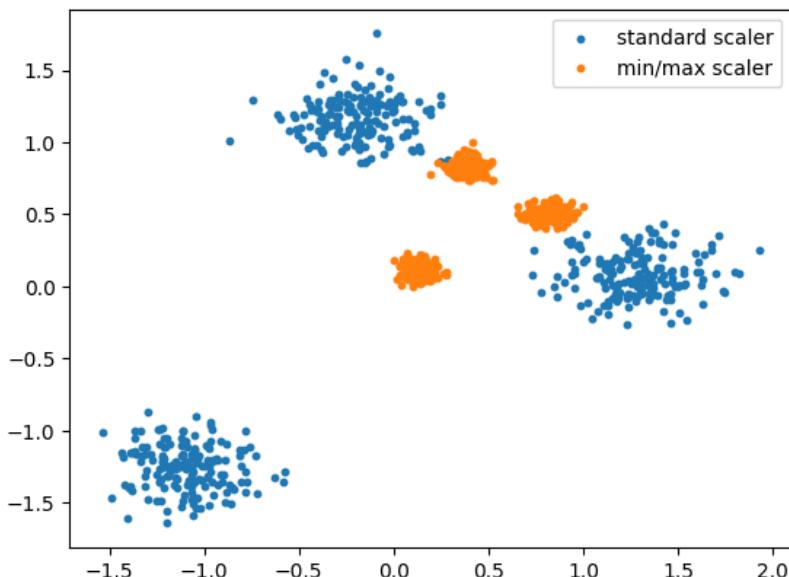


Рис.15. Сравнение стандартного масштабирования и минимаксного масштабирования.

#### 5.2.1.2.Метрики качества основанные на расстоянии

Форма и размер кластеров в алгоритма DBSCAN сильно зависят от гиперпараметра  $\epsilon$ . Все кластеризации зависят от своих гиперпараметров, поэтому их следует как-то выбирать/подбирать. В машинном обучении существует два основных способа оценки гиперпараметров – внешняя оценка и внутренняя оценка.

При внешней оценке используется эксперт, который определяет для каждого значения гиперпараметра насколько результат хорош, и выбирается то значение гиперпараметра, которое соответствует наилучшему результату с точки зрения эксперта.

При внутренней оценке рассчитывается некая метрика качества, и в зависимости от ее величины (максимальной или минимальной – зависит от конкретной метрики) при каждом значении гиперпараметра принимается решение о выборе наилучшего результата.

Для метрических методов кластеризации существует много различных метрик, обсудим наиболее широко используемую – коэффициент силуэта.

$$SILH = \frac{1}{N} \sum_{i=1}^N SILH_i$$

$$SILH_i = \frac{b_i - a_i}{\max(a_i, b_i)}$$

здесь  $a_i, b_i$  – средний размер кластера и расстояние до центра ближайшего к нему кластера соответственно.

С качественной точки зрения коэффициент силуэта – это среднее по парам кластеров отношение расстояния между ближайшими границами кластеров к расстоянию между их центрами. Когда удается разбить данные на кластера малых размеров, разнесенных далеко, коэффициент силуэта принимает свое максимальное значение, равное 1. Коэффициент силуэта основан на евклидовом расстоянии и модели кластеров в виде многомерных сфер. Пример поиска гиперпараметра методом силуэта приведен ниже

#### Код 21. Поиск гиперпараметра методом силуэта

```
from sklearn.metrics import silhouette_score
best_score,best_eps,best_clusters = -1,None,None
for eps in np.linspace(0.1, 5.0, 20):
    db = DBSCAN(eps=eps, min_samples=5).fit(X)
    labels = db.labels_
    if len(set(labels)) < 2 or (labels == -1).all(): continue
    score = silhouette_score(X, labels)
    if score > best_score:
        best_score = score
        best_eps = eps
        best_clusters = labels
plt.scatter(X[:, 0], X[:, 1], c=best_clusters, s=10)
plt.title('$\epsilon$ = '+str(best_eps))
```

Видно, что наилучшее значение Силуэта для этого датасета достигается при  $\epsilon = 1.6$ , а сами сферические кластера становятся разделимыми.

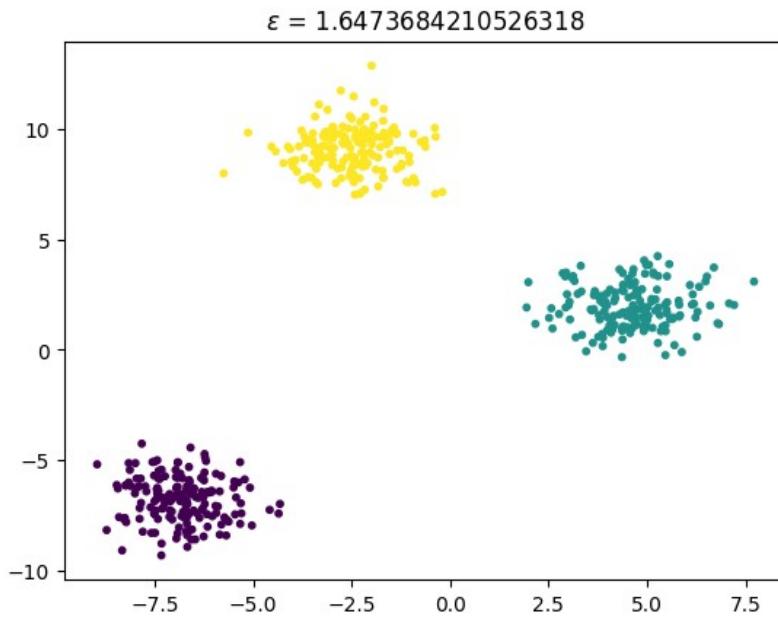


Рис.16. Поиск гиперпараметра методом силуэта

### 5.2.2.Модельные методы кластеризации.

Предположим, что у нас есть некие два генератора случайных чисел распределенных нормально, которые генерируют случайные числа. И нам надо определить для каждой точки полученных данных, какой генератор ее сгенерировал.

Как это определять? Существует метод максимального правдоподобия, который может оценить вероятность того, что точка принадлежит к тому или иному известному распределению. Исходя из этого, можно решить задачу определения параметров двух генераторов так, чтобы они наилучшим образом описывали наши данные. После того, как параметры этих генераторов определены, можно легко определить, к какому генератору наиболее вероятно относится каждая точка данных. То есть провести кластеризацию.

Таким образом, можно описывать точки, как принадлежащие к некой статистической модели, описываемой суперпозицией (суммой) нормальных распределений. Гиперпараметром в этом случае выступает количество этих распределений, а остальные параметры определяются в процессе решения задачи.

Такие методы кластеризации называются модельными, и в качестве базовых распределений могут выступать любые распределения. Наиболее часто в качестве распределений используются нормальные распределения, а метод кластеризации называется Гауссовой Смесью (Gaussian Mixture).

#### 5.2.2.1.Гауссова смесь

Многомерное нормальное распределение имеет вид:

$$f(\vec{x}|\vec{\mu}, \Sigma) = \frac{1}{(2\pi)^{n/2} |\Sigma|^{1/2}} e^{-\frac{(\vec{x}-\vec{\mu})^T \Sigma^{-1} (\vec{x}-\vec{\mu})}{2}}; \vec{x}, \vec{\mu} \in R^n, \Sigma \in R^{n \times n}$$

здесь вектор  $\vec{\mu}$  — среднее значение, а  $\Sigma$  — матрица ковариаций данного распределения. Модель плотности вероятности гауссовой смеси имеет вид:

$$p(x) = \sum_{i=1}^N A_i f(\vec{v} | \mu_i, \Sigma_i)$$

где  $N, A_{i \in [1..N]}, \mu_{i \in [1..N]}, \Sigma_{i \in [1..N]}$  - неизвестные параметры требующие определения.

Модельные алгоритмы предполагают определение множества параметров выбранной модели. Чаще всего в случае смеси распределений используется EM (Expectation-Maximization) алгоритм – итеративный алгоритм, определяющий параметры методом последовательных приближений – циклическим выполнением пары шагов – Expectation и Maximization. За счет этого алгоритм кластеризации достаточно быстр, хотя и зависит от случайных начальных условий – начальных приближений для параметров каждого распределения.

#### Код 22. Кластеризация методом гауссовой смеси

```
from sklearn.mixture import GaussianMixture
gmm = GaussianMixture(n_components=3, random_state=42)
clusters = gmm.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=clusters, s=10)
```

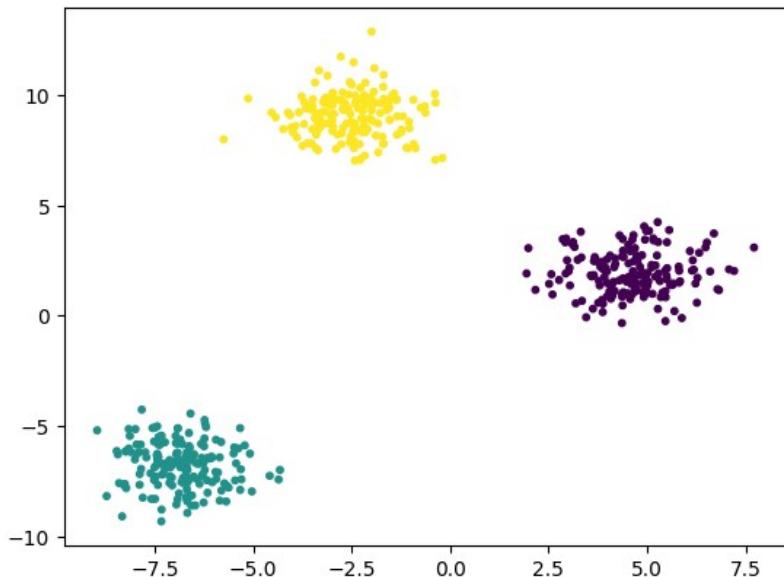


Рис.17. Кластеризация методом гауссовой смеси

Преимуществом модельных алгоритмов является то, что они могут проводить не только жесткую, но и мягкую кластеризацию и оценивать для каждой точки вероятность принадлежности к каждому классу.

#### Код 23. Работа гауссовой смеси в режиме мягкой кластеризации

```
gmm.fit(X)
clusters_prob = gmm.predict_proba(X)
clusters=np.argmax(clusters_prob, axis=-1)
plt.scatter(X[:, 0], X[:, 1], c=clusters, s=10)
```

Можно проверить, что результаты не изменились — мягкая кластеризация — основной режим работы гауссовой смеси.

Другим достоинством модельных алгоритмов является возможность кластеризовать новые неизвестные ранее точки, поскольку модель распределения точек уже построена. Это позволяет оценить границы, по которым происходит разделение точек на различные кластера.

**Код 24. Работа гауссовой смеси в режиме предсказания кластеров для новых данных**

```
X1,X2=X[:300,:],X[300:,:]
gmm.fit(X1)
clusters = gmm.predict(X2)
plt.scatter(X2[:, 0], X2[:, 1], c=clusters, s=10)
```

Можно проверить, что результаты также хорошо предсказывают кластера для новых данных из того-же распределения (домена).

Еще одним достоинством алгоритма GM является его устойчивость к изменению масштабов осей или поворотах — при таких изменениях алгоритм просто изменит параметры нормальных распределений.

### 5.2.2.2.Метрики качества основанные на вероятности

Основным гиперпараметром модельных алгоритмов является число кластеров. Для поиска этого гиперпараметра обычно используется семейство информационных критериев (IC) – Байесовский информационный критерий (BIC, или критерий Шварца SC):

$$BIC = k \ln n - 2L$$

или критерий Акаике (AIC):

$$AIC = 2k - 2L$$

где  $k$  — число параметров в статистической модели,  $L$  — значение натурального логарифма функции правдоподобия модели,  $n$  — объем выборки по которой строилась модель.

Логарифмическая функция правдоподобия определяется по датасету как вероятность того, что все точки датасета  $x_i$  принадлежат распределению с плотностью вероятности  $P(x)$ :

$$L = \ln(P(x_0)P(x_1)\dots P(x_N)) = \sum_{i=0}^N \ln(P(x_i))$$

Оптимальная модель соответствует минимуму значения выбранного информационного критерия:

**Код 25. Поиск оптимального числа GM кластеров по критерию BIC**

```
from sklearn.mixture import GaussianMixture
best_bic,best_n,best_gmm = 1e100,-1,-1
X, y = make_blobs(n_samples=500, centers=2, n_features=2, random_state=42, cluster_std=2)
fig,axs=plt.subplots(3,3,figsize=(6,6),constrained_layout=True)
for n in range(1, 10):
    gmm = GaussianMixture(n_components=n, covariance_type='full', random_state=42)
    clusters=gmm.fit_predict(X)
    bic=gmm.bic(X)
    if bic<best_bic:
        best_bic,best_n,best_gmm=bic,n,gmm
        axs[(n-1)//3,(n-1)%3].scatter(X[:, 0], X[:, 1], c=clusters, s=10,cmap='tab10')
```

```

axs[(n-1)//3,(n-1)%3].set_title(str(n)+' clusters')
axs[(best_n-1)//3,(best_n-1)%3].set_title('The best:'+str(best_n)+' clusters')

```

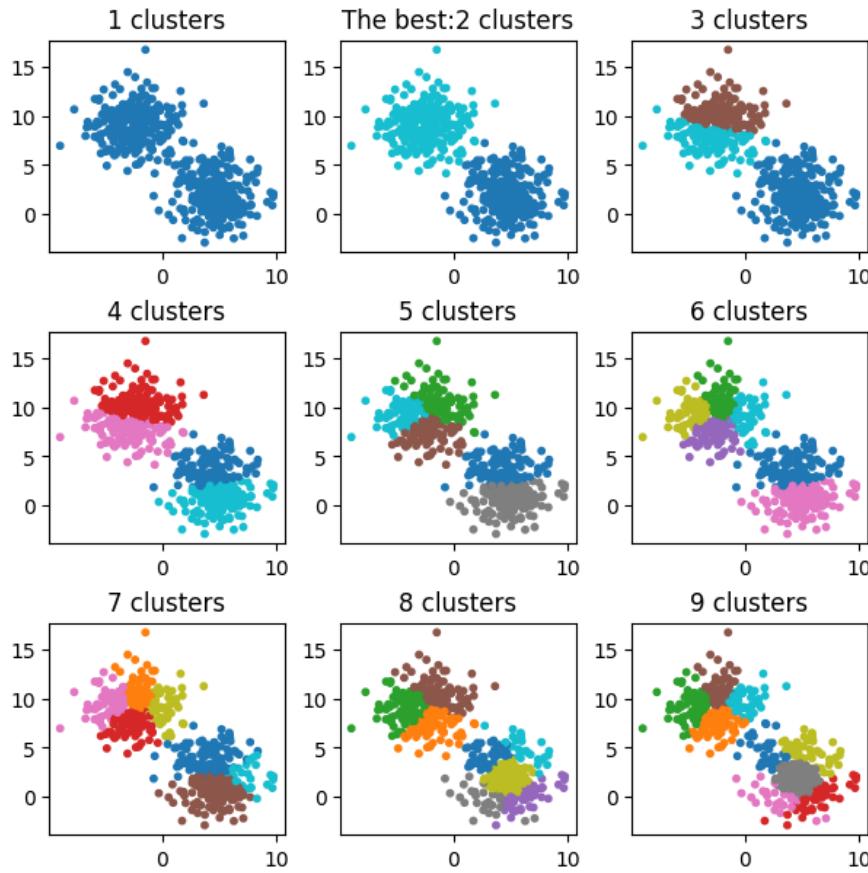


Рис.18. Поиск оптимального числа GM кластеров по критерию BIC

## 5.3.Иерархические алгоритмы

Недостатком многих описанных метрических методов является сильная зависимость результатов от значения гиперпараметра. Сделать такую зависимость плавной помогают иерархические алгоритмы.

Иерархический метод строит сразу некий набор разделений, непрерывно зависящий от гиперпараметра. Таким образом при небольшом изменении гиперпараметра разбиение изменится несильно. Обычно в качестве гиперпараметра выступает количество кластеров, и при его небольшом изменении только несколько кластеров меняют свою форму, остальные кластера оказываются неизменными.

К иерархическим методам относят агglomerативную и дивизивную кластеризацию.

### 5.3.1.Агglomerативная кластеризация

Агglomerативная кластеризация строит кластеризацию по убыванию числа кластеров. При агglomerативной кластеризации вначале предполагается, что каждая точка датасета находится в собственном кластере. На каждом следующем шаге находятся два самых

близких кластера и они объединяются в один. Иттерации повторяются пока не останется заданное число кластеров (или один кластер). Таким образом, каждая кластеризация отличается от предыдущей только двумя кластерами. Гиперпараметром при такой кластеризации выступает количество кластеров, на котором мы остановились.

#### Код 26. Пример аггломеративной кластеризации

```
from sklearn.cluster import AgglomerativeClustering
fig,axs=plt.subplots(3,3,figsize=(6,6),constrained_layout=True)
for n in range(1,10):
    agg = AgglomerativeClustering(n_clusters=n, metric='euclidean')
    clusters = agg.fit_predict(X)
    axs[(n-1)//3,(n-1)%3].scatter(X[:, 0], X[:, 1], c=clusters, s=10,cmap='tab10')
    axs[(n-1)//3,(n-1)%3].set_title(str(n)+' clusters')
```

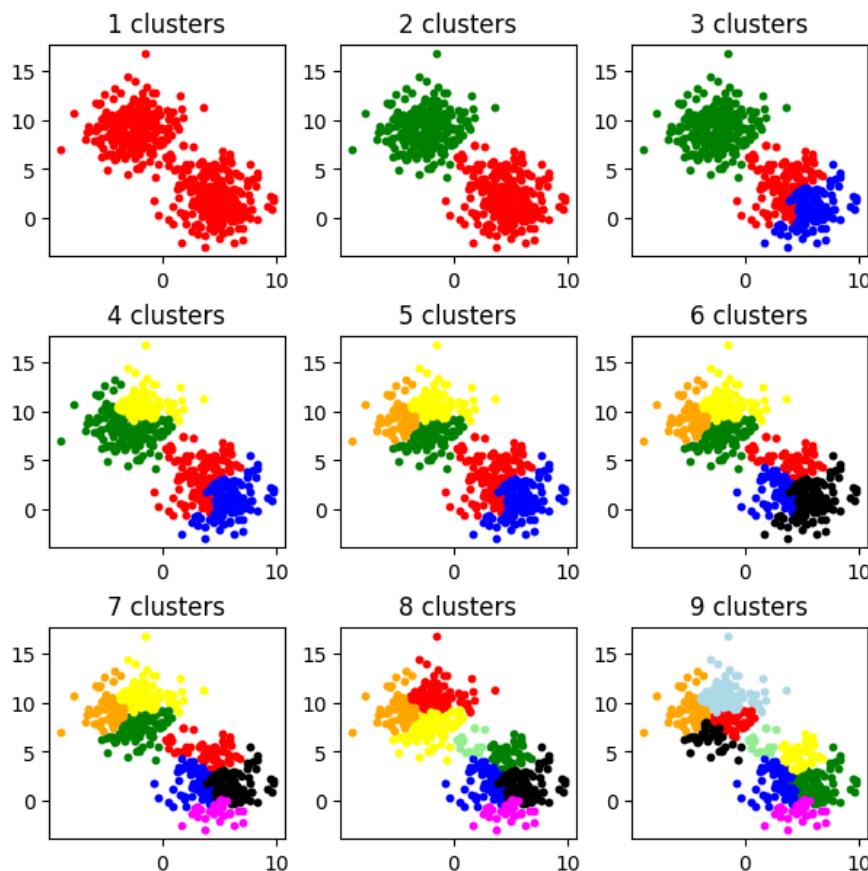


Рис.19. Пример аггломеративной кластеризации

Одним из методов аггломеративной кластеризации является кластеризация минимальным остовным деревом (MST-based clustering).

### 5.3.2.Дивизивная кластеризации

Дивизивная кластеризация строит разделения по увеличению числа кластеров. При дивизивной кластеризации вначале предполагается, что все точки датасета находятся в

одном кластере. На каждом следующем шаге находится самый большой кластер и он разделяется на два. Итерации повторяются пока не останется заданное число кластеров (или все точки будут расположены в уникальных кластерах). Таким образом, каждая кластеризация отличается от предыдущей только двумя кластерами. Гиперпараметром при такой кластеризации выступает количество кластеров, на котором мы остановились.

#### Код 27. Пример дивизивной кластеризации

```
from sklearn.cluster import KMeans
def divisive_clustering(data, labels, depth=0):
    if len(labels)==0: labels=[0]*len(data)
    if depth == 0 or len(data)<2: return data,labels,0
    else:
        dataP,labelsP,sm=divisive_clustering(data, labels, depth=depth-1)
        datan,labelsn=np.array(dataP),np.array(labelsP)
        s_max,l_max=-1,-1
        for l in set(labelsP):
            data1=datan[labelsn==l]
            s=data1.std(axis=0).max()
            if s>s_max: s_max,l_max=s,l
        dataout,labelsout=[],[]
        for l in set(labelsP):
            if(l!=l_max):
                dataout+=list(datan[labelsn==l])
                labelsout+=[l]*len(list(datan[labelsn==l]))
            else:
                if(datan[labelsn==l].shape[0]>2):
                    kmeans=KMeans(n_clusters=2,random_state=42).fit(datan[labelsn==l])
                    dataout+=list(datan[labelsn==l])
                    labelsout+=list(np.array(kmeans.labels_)+sm+1)
                else:
                    dataout+=list(datan[labelsn==l])
                    labelsout+=[l]*len(list(datan[labelsn==l]))
        return dataout,labelsout,max(labelsout)

fig,axs=plt.subplots(3,3,figsize=(6,6),constrained_layout=True)
for n in range(1,10):
    data,clusters,sm = divisive_clustering(X, [], depth=n);
    data=np.array(data)
    axs[(n-1)//3,(n-1)%3].scatter(data[:, 0], data[:, 1], c=clusters, s=10,cmap='tab10')
    axs[(n-1)//3,(n-1)%3].set_title(str(n)+' clusters')
```

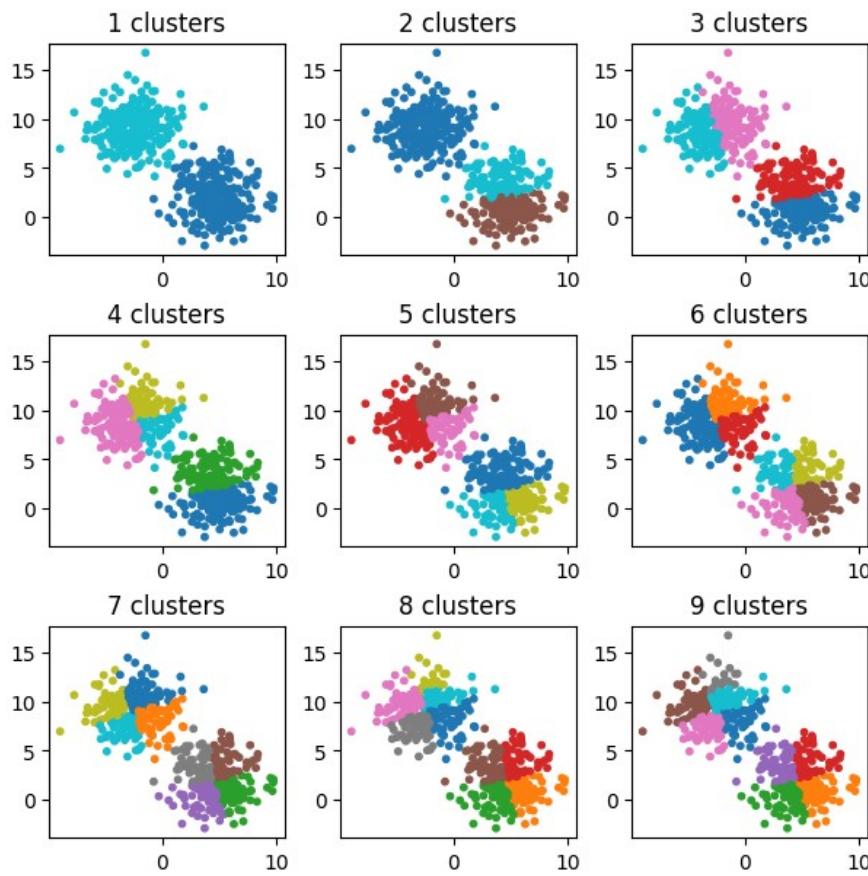


Рис.20. Пример дивизивной кластеризации

### 5.3.3.Метрики качества

Поскольку при иерархических методах кластеризации обычно используется расстояние, то в качестве метрик качества для поиска гиперпараметров удобно выбирать уже известные нам метрические метрики качества, например коэффициент Силуэта. Или использовать внешнюю оценку качества экспертом.

## 5.4.Смешанные алгоритмы

Недостатком модельных методов является простота форм кластеров, которые они способны обнаружить. В этом смысле они значительно сложнее метрических методов, которые способны выделять кластера сложной формы. Недостатком метрических алгоритмов является невозможность предсказывать кластера новых данных и делать мягкую кластеризацию.

Одним из интересных семейств методов являются смешанные алгоритмы, которые используют несколько методов кластеризации одновременно – чаще всего модельные и метрические одновременно. Такие методы позволяют объединить положительные стороны обоих типов алгоритмов. К таким методам относятся например GMsDB, DBSCAN-GM и подобные им.

В основе метода GMsDB лежит разбиение кластеров методом GM с оценкой числа гауссовых кластеров исходя из критерия BIC, а затем объединение сильно пересекающихся

кластеров. Это позволяет строить модели кластеров более сложной формы, чем эллиптические кластера GM. Метод статистический, поэтому недостатком является требование большого количества точек для кластеризации.

Код 28. Пример кластеризации смешанным методом GMsDB

```
from gmsdb import GMSDB
from sklearn.datasets import make_moons
X, y = make_moons(n_samples=3000, random_state=42, noise=0.05)
cl=GMSDB(n_components=50)
clusters=cl.fit_predict(X)
plt.scatter(X[:, 0], X[:, 1], c=clusters, s=10);
```

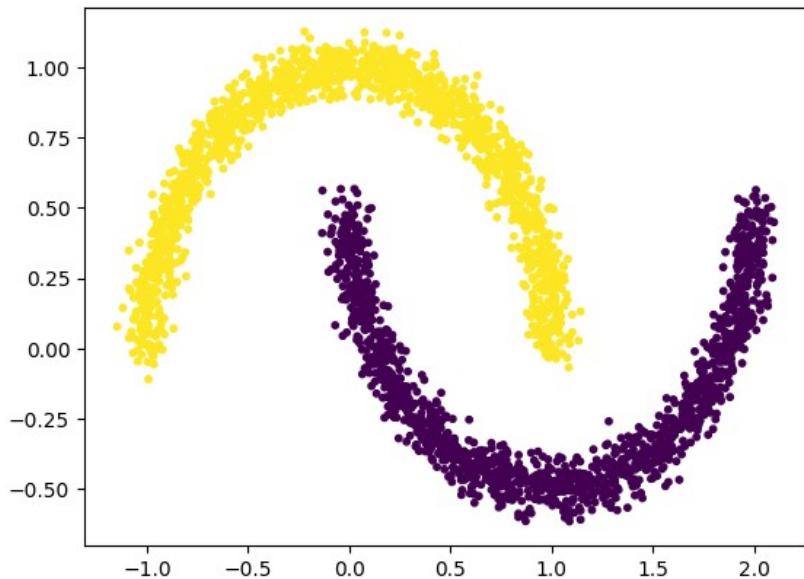


Рис.21. Пример кластеризации смешанным методом GMsDB

## **6.Проверка статистических гипотез**

### **6.1.Введение**

Допустим вы измеряете длину спички. Даже если вы измеряете точно, например микрометром, те данные которые вы измерили не всегда совпадают с вашими ожиданиями и гарантиями производителя. То есть на самом деле вы всегда работаете с какими-то случайными данными. Возникает вопрос: величина которую вы измерили соответствует стандартам или нет? Для того, чтобы формализовать задачу, вам необходимо указывать доверительный интервал, при попадании в который вы считаете измерения соответствующими стандарту. Использование доверительного интервала позволяют единым образом указывать ту точность которая соответствует измеряемым вами параметрам.

Давайте рассмотрим интервальные оценки. Допустим измеряемая нами величина принадлежит к некому известному нам распределению. Определить, принадлежит новое значение к этому распределению или нет – достаточно просто: если она не попадает в доверительный интервал уровня  $p$ , то вероятность того, что она принадлежит этому распределению не более ( $1-p$ ). Можно говорить, что исходную гипотезу о принадлежности этой величины к заданному распределению можно отбросить с уровнем значимости  $\alpha = 1 - p$ .

Обычно в статистике если специально не оговаривается, используются уровни значимости 0.05 или 95% доверительные интервалы.

Для нормальных распределений можно ввести нормированную ошибку, поделив разницу между средним и измеренным значением на среднеквадратичное отклонение. Очевидно, что для вхождения величины в 95% доверительный интервал необходимо, чтобы модуль получившегося параметра лежал в пределах [-2,2] вне зависимости от характеристик этого нормального распределения.

Решение подобных задач сводится таким образом к проверке статистических гипотез.

Для проверки статистических гипотез с помощью заданного критерия всегда необходимо три составляющих.

Первая составляющая – это нулевая ( $H_0$ ) и альтернативная ( $H_1$ ) гипотезы. При проверке статистических гипотез чаще всего мы проверяем – можно отвергнуть нулевую гипотезу (в пользу альтернативной) или нет. Гипотезы должны быть взаимоисключающими, и отвержение основной гипотезы автоматически приводит к принятию альтернативной. Важное замечание – проверка статистических гипотез не может доказать выполнение нулевой гипотезы, она может лишь проверить возможность ее опровержения с использованием заданного критерия.

Вторая составляющая – статистика. Статистика – это любая функция от случайных величин. Поэтому статистика сама является случайной величиной, принадлежащей некому распределению.

Третья составляющая – вид распределения этой статистики при выполнении основной гипотезы  $H_0$ . Обычно этот вид либо определяется из теоретических соображений, либо табулируется в результате численных расчетов.

Обычно критерии делятся на две большие группы – параметрические и непараметрические критерии. К параметрическим критериям относятся критерии, статистики которых зависят только от выборочных средних и дисперсий случайных величин участвующих в анализе. Остальные критерии относят к непараметрическим.

Еще одним разделением критериев является разделение на Z и T критерии. К Z критериям относятся критерии, для которых распределение статистики при нулевой гипотезе полностью известно, а к T-критериям – те критерии, для которых параметры распределения нужно оценивать из самих анализируемых данных.

Еще одним разделением критериев является разделение на одновыборочные и

двуихвыборочные критерии, в зависимости от того нужно определять параметры какого-то распределения или сравнивать два распределения.

Двуихвыборочные критерии в свою очередь делятся на связанные и несвязанные выборки в зависимости от того, можно однозначно идентифицировать как изменилось значение каждого элемента выборки или нет.

Следует заметить, что каждый критерий обладает своей мощностью – величиной от 0 до 1, отражающей вероятность верного принятия альтернативной гипотезы. Чем выше мощность критерия — тем он точнее и более рекомендуется к использованию.

Кроме того, как и доверительные интервалы, критерии делятся на двухсторонние, левосторонние и правосторонние в зависимости от типа используемого доверительного интервала для распределения статистики при нулевой гипотезе.

Детально можно посмотреть работы [4,5].

## 6.2.Параметрические критерии

### 6.2.1.Одновыборочные критерии

#### 6.2.1.1.Z-критерий для среднего

Одновыборочный Z-критерий применяется для проверки гипотезы о том, что среднее значение ( $\mu$ ) некой генеральной совокупности данных, из которых выбраны ваши данные длиной  $n$  равно некоторому заданному значению  $\mu_0$ , при условии, что данные имеют нормальное распределение (или выборка достаточно большая,  $n \geq 30$ ) и известно стандартное отклонение генеральной совокупности ( $\sigma$ ).

Таблица 1. Определения для одновыборочного Z-критерия:

Нулевая гипотеза ( $H_0$ )	$\mu = \mu_0$
Альтернативная гипотеза ( $H_1$ )	$\mu \neq \mu_0$ -двуихсторонняя гипотеза $\mu < \mu_0$ -левосторонняя гипотеза $\mu > \mu_0$ -правосторонняя гипотеза
Статистика	$Z = \frac{\bar{X} - \mu_0}{\sigma / \sqrt{n}}$
Нулевое распределение	$Z \sim N(0, 1)$
p-value	$p_v = 2(1 - F( Z ))$ - двуихсторонняя $p_v = (1 - F(Z))$ - правосторонняя $p_v = F(Z)$ - левосторонняя здесь $F$ — функция распределения для заданной плотности вероятности нулевого распределения

Код 29. Проверка одновыборочного Z-критерия

```

import math
import scipy.stats as ss
N=100
observed = ss.norm(1.1,1).rvs(N)
Xm=observed.mean()
Mu=1
Std=1
Z=(Xm-Mu)/(Std/np.sqrt(N))
print('pvalue two-sided',(1-ss.norm.cdf(abs(Z)))*2.)
print('pvalue right-sided',(1-ss.norm.cdf(Z)))

#####
pvalue two-sided 0.8385
pvalue right-sided 0.4192

```

Видно, что p-value больше уровня значимости 0.05, поэтому нулевая гипотеза о равенстве средних не отвергается.

### 6.2.1.2. Т-критерий для среднего

Используется для проверки гипотезы о равенстве среднего генеральной совокупности  $\mu$  заданному значению  $\mu_0$  при неизвестном стандартном отклонении  $s$  и длине выборки  $n$ . Условия применения: данные имеют нормальное распределение (или выборка достаточно большая,  $n \geq 30$ );  $\sigma$  неизвестно (вместо него используется выборочное стандартное отклонение  $s$ ).

Таблица 2. Определения для одновыборочного Т-критерия:	
Нулевая гипотеза ( $H_0$ )	$\mu = \mu_0$
Альтернативная гипотеза ( $H_1$ )	$\mu \neq \mu_0$ -двухсторонняя гипотеза $\mu < \mu_0$ -левосторонняя гипотеза $\mu > \mu_0$ -правосторонняя гипотеза
Статистика	$T = \frac{\bar{X} - \mu_0}{s/\sqrt{n}}$
Нулевое распределение	$T \sim t(n-1)$ - распределение Стьюдента с числом степеней свободы $n-1$
p-value	$p_v = 2(1 - F( T ))$ - двухсторонняя $p_v = (1 - F(T))$ - правосторонняя $p_v = F(T)$ - левосторонняя

Код 30. Проверка одновыборочного Т-критерия

```

Xm=observed.mean()
Mu=1
Std=observed.std()
T=(Xm-Mu)/(Std/math.sqrt(N))
pvalTS=2*(1-ss.t.cdf(abs(T),df=observed.shape[0]-1))
pvalRS=(1-ss.t.cdf(T,df=observed.shape[0]-1))
print('pvalue two-sided',pvalTS)
print('pvalue right-sided',pvalRS)

#####
pvalue two-sided 0.8485
pvalue right-sided 0.5757

```

Видно, что p-value больше уровня значимости 0.05, поэтому нулевая гипотеза о равенстве средних не отвергается.

### **6.2.1.3. Z-критерий для доли**

Используется для проверки гипотезы о равенстве доли признака в генеральной совокупности  $p$  заданному значению  $p_0$  при длине выборки  $n$ . Условия применения: выборка достаточно большая (обычно  $np_0 \geq 10$  и  $n(1-p_0) \geq 10$ ), а данные имеют биномиальное распределение, аппроксимируемое нормальным.

Таблица 3. Определения для одновыборочного Z-критерия для доли:	
Нулевая гипотеза ( $H_0$ )	$p = p_0$
Альтернативная гипотеза ( $H_1$ )	$p \neq p_0$ -двухсторонняя гипотеза $p < p_0$ -левосторонняя гипотеза $p > p_0$ -правосторонняя гипотеза
Статистика	$Z = \frac{p - p_0}{\sqrt{p_0(1-p_0)/n}}$
Нулевое распределение	$Z \sim N(0, 1)$
p-value	$p_v = 2(1 - F( Z ))$ - двухсторонняя $p_v = (1 - F(Z))$ - правосторонняя $p_v = F(Z)$ - левосторонняя

Код 31. Проверка одновыборочного Z-критерия для доли

```

N=100
observed = ss.bernoulli(0.3).rvs(N)
p=observed.mean()
p0=0.31
Z=(p-p0)/(np.sqrt(p0*(1-p0)/N))
print('pvalue two-sided',(1-ss.norm.cdf(abs(Z)))*2.)
print('pvalue right-sided',(1-ss.norm.cdf(Z)))

#####
pvalue two-sided 0.8288
pvalue right-sided 0.4144

```

Видно, что p-value больше уровня значимости 0.05, поэтому нулевая гипотеза о равенстве долей не отвергается.

## 6.2.2. Двухвыборочные критерии для несвязанных выборок

### 6.2.2.1. Z-критерий для среднего

Используется для сравнения средних значений двух независимых генеральных совокупностей ( $\mu_1$  и  $\mu_2$ ) при известных стандартных отклонениях ( $\sigma_1$  и  $\sigma_2$ ) по выборкам длиной  $n_1$  и  $n_2$  соответственно. Условия применения: независимые выборки, нормальное распределение данных (или  $n_1, n_2 \geq 30$ ), известные  $\sigma_1$  и  $\sigma_2$ .

Таблица 4. Определения для двухвыборочного Z-критерия для среднего (несвязанные выборки):

Нулевая гипотеза ( $H_0$ )	$\mu_1 = \mu_2$
Альтернативная гипотеза ( $H_1$ )	$\mu_1 \neq \mu_2$ -двухсторонняя гипотеза $\mu_1 < \mu_2$ -левосторонняя гипотеза $\mu_1 > \mu_2$ -правосторонняя гипотеза
Статистика	$Z = \frac{\mu_1 - \mu_2}{\sqrt{\frac{\sigma_1^2}{n_1} + \frac{\sigma_2^2}{n_2}}}$
Нулевое распределение	$Z \sim N(0, 1)$
p-value	$p_v = 2(1 - F( Z ))$ - двухсторонняя $p_v = (1 - F(Z))$ - правосторонняя $p_v = F(Z)$ - левосторонняя

Код 32. Проверка двухвыборочного Z-критерия для среднего (несвязанные выборки)

```

N1=100
N2=100
observed1 = ss.norm(1.1,1).rvs(N1)
observed2 = ss.norm(1.0,1).rvs(N2)
Mu1=observed1.mean()
Mu2=observed2.mean()
Std1=1
Std2=1
Z=(Mu1-Mu2)/np.sqrt(Std1**2/np.sqrt(N1)+Std2**2/np.sqrt(N2))
print('pvalue two-sided',(1-ss.norm.cdf(abs(Z)))*2.)
print('pvalue right-sided',(1-ss.norm.cdf(Z)))

#####
pvalue two-sided 0.5401
pvalue right-sided 0.2700

```

Видно, что p-value больше уровня значимости 0.05, поэтому нулевая гипотеза о равенстве средних не отвергается.

### 6.2.2.2. Т-критерий Уэлча для среднего

Используется для сравнение средних двух независимых генеральных совокупностей ( $\mu_1, \mu_2$ ) при неизвестных дисперсиях ( $s_1^2, s_2^2$ ), по выборкам длиной  $n_1, n_2$  соответственно.

Таблица 5. Определения для двухвыборочного Т-критерия Уэлча для среднего (несвязанные выборки):

Нулевая гипотеза ( $H_0$ )	$\mu_1 = \mu_2$
Альтернативная гипотеза ( $H_1$ )	$\mu_1 \neq \mu_2$ -двухсторонняя гипотеза $\mu_1 < \mu_2$ -левосторонняя гипотеза $\mu_1 > \mu_2$ -правосторонняя гипотеза
Статистика	$T = \frac{\mu_1 - \mu_2}{\sqrt{s_1^2/n_1 + s_2^2/n_2}}$
Нулевое распределение	$T \sim t(df)$ $df = \frac{(s_1^2/n_1 + s_2^2/n_2)^2}{(s_1^2/n_1)^2/(n_1-1) + (s_2^2/n_2)^2/(n_2-1)}$
p-value	$p_v = 2(1 - F( T ))$ - двухсторонняя $p_v = (1 - F(T))$ - правосторонняя $p_v = F(T)$ - левосторонняя

Важное замечание (проблема Беренца-Фишера): эту задачу можно решить только если  $n_1 \geq n_2$  при  $s_1 \geq s_2$ . Решение проблемы – стараться использовать выборки одинаковой длины или всегда проверять указанный критерий.

Код 33. Проверка двухвыборочного Т-критерия Уэлча для среднего (несвязанные выборки)

```

N1,N2=100,200
observed1 = ss.norm(1.1,1).rvs(N1)
observed2 = ss.norm(1.0,2).rvs(N2)
Mu1=observed1.mean()
Mu2=observed2.mean()
S1=observed1.std()
S2=observed2.std()
df = ((S1**2/N1)**2 + (S2**2/N2)**2) / ((S1**2/N1)**2/(N1-1) + (S2**2/N2)**2/(N2-1))
T=(Mu1-Mu2)/np.sqrt(Std1**2/np.sqrt(N1)+Std2**2/np.sqrt(N2))
print('pvalue two-sided',(1-ss.t(df).cdf(abs(T)))*2.)
print('pvalue right-sided',(1-ss.t(df).cdf(T)))
print('df:',df)

#####
pvalue two-sided 0.7988
pvalue right-sided 0.3994
df: 162.9645

```

Видно, что p-value больше уровня значимости 0.05, поэтому нулевая гипотеза о равенстве средних не отвергается.

### **6.2.2.3.Z-критерий для доли**

Используется для сравнения долей признака  $p_1, p_2$  в двух независимых генеральных совокупностях по выборкам длиной  $n_1, n_2$ . Условия применения: независимые выборки,  $n_1 p_1 \geq 10, n_1 (1 - p_1) \geq 10, n_2 p_2 \geq 10, n_2 (1 - p_2) \geq 10$

Таблица 6. Определения для двухвыборочного Z-критерия для доли (несвязанные выборки):	
Нулевая гипотеза ( $H_0$ )	$p_1 = p_2$
Альтернативная гипотеза ( $H_1$ )	$p_1 \neq p_2$ -двухсторонняя гипотеза $p_1 < p_2$ -левосторонняя гипотеза $p_1 > p_2$ -правосторонняя гипотеза
Статистика	$p_0 = \frac{p_1 n_1 + p_2 n_2}{n_1 + n_2}$ $Z = \frac{p_1 - p_2}{\sqrt{p_0(1-p_0)(1/n_1+1/n_2)}}$
Нулевое распределение	$Z \sim N(0, 1)$
p-value	$p_v = 2(1 - F( Z ))$ - двухсторонняя $p_v = (1 - F(Z))$ - правосторонняя $p_v = F(Z)$ - левосторонняя

Код 34. Проверка двухвыборочного Z-критерия для доли (несвязанные выборки)

```

observed1 = ss.bernoulli(0.45).rvs(N1)
observed2 = ss.bernoulli(0.42).rvs(N2)
p1=observed1.mean()
p2=observed2.mean()
p0=(p1*N1+p2*N2)/(N1+N2)
Z=(p1-p2)/(np.sqrt(p0*(1-p0)*(1/N1+1/N2)))
print('pvalue two-sided',(1-ss.norm.cdf(abs(Z)))*2.)
print('pvalue right-sided',(1-ss.norm.cdf(Z)))
#####
pvalue two-sided 0.2491
pvalue right-sided 0.1245

```

Видно, что p-value больше уровня значимости 0.05, поэтому нулевая гипотеза о равенстве долей не отвергается.

### 6.2.3. Двухвыборочные критерии для связанных выборок

#### 6.2.3.1. Т-критерий для средних

Используется для сравнения средних значений одной и той же группы объектов в двух  $X_{1i}, X_{2i}$  разных условиях (до/после воздействия) или для естественно парных наблюдений по выборкам длины n.

Таблица 7. Определения для двухвыборочного Т-критерия для среднего (связанные выборки):

Нулевая гипотеза ( $H_0$ )	$\mu_1 = \mu_2$
Альтернативная гипотеза ( $H_1$ )	$\mu_1 \neq \mu_2$ -двухсторонняя гипотеза $\mu_1 < \mu_2$ -левосторонняя гипотеза $\mu_1 > \mu_2$ -правосторонняя гипотеза
Статистика	$d_i = X_{1i} - X_{2i}$ $\bar{d} = \frac{\sum d_i}{n}$ $s_d = \sqrt{\frac{\sum (d_i - \bar{d})^2}{n-1}}$ $T = \frac{\bar{d}}{s_d / \sqrt{n}}$
Нулевое распределение	$T \sim t(n-1)$
p-value	$p_v = 2(1 - F( T ))$ - двухсторонняя $p_v = (1 - F(T))$ - правосторонняя $p_v = F(T)$ - левосторонняя

Код 35. Проверка двухвыборочного Т-критерия для среднего (связанные выборки):

```

observed1 = ss.norm().rvs(N1)
observed2 = observed1+ss.norm().rvs(N1)*0.1+0.02
t_stat, p_value = ss.ttest_rel(observed1, observed2, alternative='two-sided')
print('p_value two-sided',p_value)
###
p_value two-sided 0.1748

```

Видно, что p-value больше уровня значимости 0.05, поэтому нулевая гипотеза о равенстве средних не отвергается.

### 6.2.3.2.Z-критерий для доли

Этот критерий для доли (модифицированный критерий Макнемара с поправкой Эдвардса) используется для проверки изменения доли одной и той же группы объектов в двух связанных условиях (например, до/после воздействия) для бинарных данных по выборкам длины n.

Таблица 8. Определения для двухвыборочного Z-критерия для доли (связанные выборки):

Нулевая гипотеза ( $H_0$ )	$p_1 = p_2$
Альтернативная гипотеза ( $H_1$ )	$p_1 \neq p_2$ -двухсторонняя гипотеза $p_1 < p_2$ -левосторонняя гипотеза $p_1 > p_2$ -правосторонняя гипотеза
Статистика	Матрица сопряженности: $\begin{pmatrix} a & b \\ c & d \end{pmatrix}$ , $b+c \geq 5$ $Z = \frac{b-c}{\sqrt{(b+c)-\frac{(b-c)^2}{n}}}$
Нулевое распределение	$Z \sim N(0, 1)$
p-value	$p_v = 2(1 - F( Z ))$ - двухсторонняя $p_v = (1 - F(Z))$ - правосторонняя $p_v = F(Z)$ - левосторонняя

Код 36.Проверка двухвыборочного Т-критерия для среднего (связанные выборки):

```

table = [[90,20],[10, 80]]
from statsmodels.stats.contingency_tables import mcnemar
result = mcnemar(table)
print("p-value two-sided:",float(result.pvalue))
###
p-value two-sided: 0.0987

```

Видно, что p-value больше уровня значимости 0.05, поэтому нулевая гипотеза о равенстве долей не отвергается.

## 6.3. Непараметрические критерии

Чаще всего параметрические критерии требуют: нормальности распределения, однородности дисперсий, интервальной/отношений шкалы измерений. Кроме того чаще всего критерии работают только со средними, долями или дисперсиями. Для решения более сложных или более общих задач они не подходят, и используют параметрические критерии. Основу множества параметрических критериев составляет перестановка значений. Некоторые параметрические критерии можно использовать для работы с категориальными данными. Непараметрические тесты обычно работают с квантильными оценками, используют ранги исходных значений, что снижает влияние аномалий и выбросов на результаты.

Таблица 9. Соответствие параметрических и непараметрических критериев		
Параметрический критерий	Непараметрический критерий	Условие выбора
Двухвыборочный t-тест	Манна-Уитни	Ненормальные данные или разные дисперсии
Парный t-тест	Уилкоксон (парный)	Маленькая выборка или порядковые данные
ANOVA	Крускала-Уоллиса	>2 группы с неравными дисперсиями
-	Критерий знаков	Крайне малые выборки ( $n < 5$ )

Рассмотрим некоторые мощные непараметрические критерии.

### 6.3.1. Одновыборочный критерий Уилкоксона

Одновыборочный критерий Уилкоксона (критерий знаковых рангов Уилкоксона) предназначен для проверки гипотезы о равенстве медианы распределения в случае одномерной выборки заданному значению  $\mu_0$  из непрерывного симметричного распределения по выборке длиной  $n$ .

Таблица 10. Определения для одновыборочного критерия Уилкоксона:	
Нулевая гипотеза ( $H_0$ )	$\mu = \mu_0$
Альтернативная гипотеза ( $H_1$ )	$\mu \neq \mu_0$ -двухсторонняя гипотеза $\mu < \mu_0$ -левосторонняя гипотеза $\mu > \mu_0$ -правосторонняя гипотеза
Статистика	Вычисляем разности $d_i = x_i - \mu_0$ Исключаем нулевые разности Рангируем абсолютные значения разностей $ d_i $ . Присваиваем знаки разностей рангам. Суммируем ранги с положительными разностями: $T = \sum_{d_i > 0} rank( d_i ) sign(d_i)$

Нулевое распределение	$T \sim N\left(0, \sqrt{\frac{n(n+1)(2n+1)}{6}}\right)$
p-value	$p_v = 2(1 - F( T ))$ - двухсторонняя $p_v = (1 - F(T))$ - правосторонняя $p_v = F(T)$ - левосторонняя

Код 37. Проверка двухвыборочного Т-критерия для среднего (связанные выборки):
<pre>data = (np.random.rand(100)-0.5)+6.98 w, pvalue = ss.wilcoxon(data-7,alternative='two-sided') print('p-value two-sided:',pvalue) ###</pre> <p>p-value two-sided: 0.1356</p>

Видно, что p-value больше уровня значимости 0.05, поэтому нулевая гипотеза о равенстве медиан не отвергается.

### 6.3.2. Двухвыборочный критерий Мана-Уитни (несвязанные выборки)

Двухвыборочный критерий Мана-Уитни (U-критерий) используется для проверки гипотезы о том, что две независимые выборки длиной  $n$  и  $m$  происходят из одного распределения. Проверяется гипотеза о том, что выборки взяты из генеральных совокупностей с одинаковыми распределениями (или с одинаковыми медианами, если распределения симметричны).

Таблица 11. Определения для двухвыборочного критерия Мана-Уитни (U-критерия):	
Нулевая гипотеза ( $H_0$ )	$\mu_1 = \mu_2$
Альтернативная гипотеза ( $H_1$ )	$\mu_1 \neq \mu_2$ -двуихсторонняя гипотеза $\mu_1 < \mu_2$ -левосторонняя гипотеза $\mu_1 > \mu_2$ -правосторонняя гипотеза
Статистика	<ul style="list-style-type: none"> <li>- Объединяем две выборки <math>X = [x_1, \dots, x_n]</math> <math>Y = [y_1, \dots, y_m]</math> в один ранжированный ряд.</li> <li>- Присваиваем ранги (от 1 до <math>n+m</math>), учитывая средние ранги при наличии совпадений</li> <li>- Вычисляем сумму рангов для первой выборки:</li> </ul> $R_x = \sum_{i=1}^n rank( x_i )$ $U_x = nm + \frac{n(n+1)}{2} - R_x$ $U_y = nm - U_x$ $U = \min(U_x, U_y)$
Нулевое распределение	$U \sim N\left(\mu = \frac{nm}{2}, \sigma = \frac{\sqrt{(nm(n+m+1))}}{12}\right)$ при $n, m > 10$

p-value	$p_v = 2(1 - F( U - nm/2 ))$ - двухсторонняя (распределение U симметрично относительно $nm/2$ ) $p_v = (1 - F(U))$ - правосторонняя $p_v = F(U)$ - левосторонняя
---------	--

Код 38. Проверка двухвыборочного критерия Мана-Уитни (несвязанные выборки):

```
from scipy.stats import mannwhitneyu
data2 = (np.random.rand(100)-0.5)+7.01
MW, p = mannwhitneyu(data, data2)
print('p-value two-sided:',p)
###
p-value two-sided: 0.4335
```

Видно, что p-value больше уровня значимости 0.05, поэтому нулевая гипотеза о равенстве медиан не отвергается.

### 6.3.3. Двухвыборочный критерий Уилкоксона (связанные выборки)

Используется для проверки гипотезы о сдвиге медианы в парных (связанных) выборках длины  $n$  из непрерывного симметричного распределения. Нулевая гипотеза - медиана разностей пар наблюдений равна нулю.

Таблица 12. Определения для двухвыборочного критерия Уилкоксона:

Нулевая гипотеза ( $H_0$ )	$\mu_1 = \mu_2$
Альтернативная гипотеза ( $H_1$ )	$\mu_1 \neq \mu_2$ -двуихсторонняя гипотеза $\mu_1 < \mu_2$ -левосторонняя гипотеза $\mu_1 > \mu_2$ -правосторонняя гипотеза
Статистика	Вычисляем разности $d_i = x_i - y_i$ Исключаем нулевые разности Рангируем абсолютные значения разностей $ d_i $ . Присваиваем знаки разностей рангам. Суммируем ранги с положительными разностями: $T = \sum_{d_i > 0} rank( d_i ) sign(d_i)$
Нулевое распределение	$T \sim N\left(\mu = 0, \sigma = \sqrt{\frac{n(n+1)(2n+1)}{6}}\right)$
p-value	$p_v = 2(1 - F( T ))$ - двухсторонняя $p_v = (1 - F(T))$ - правосторонняя $p_v = F(T)$ - левосторонняя

Код 39. Проверка двухвыборочного критерия Уилкоксона (связанные выборки):

```

data2 = data + (np.random.rand(100)-0.5)+0.05
w, pvalue = ss.wilcoxon(data-data2,alternative='two-sided')
print('p-value two-sided:',pvalue)
###
p-value two-sided: 0.3806

```

Видно, что p-value больше уровня значимости 0.05, поэтому нулевая гипотеза о равенстве медиан не отвергается.

## 6.4.Бутстррап

Метод бутстрапа является одним из универсальных методов проверки гипотез. Мощность его невелика, но компенсируется универсальностью. Он может использоваться для определения того, принадлежит какая-то статистика к своему нулевому распределению или нет.

В качестве нулевого распределения выбранной статистики выбирается распределение исследуемого параметра по выборкам, составленным из выборок из оригинальной выборки одинаковой с ней длины с возвратом.

Код 40. Проверка одновыборочной гипотезы о медиане бутстррапом:

```

data=ss.norm.rvs(7,1,size=100)
N=data.shape[0]
Q,hq=[],[]
for i in range(100000):
    idx=np.random.randint(0,N,N)
    data1=data[idx]
    Q0=np.median(data1)
    hq.append(Q0)
print('Conf.int.95%:',float(np.percentile(hq,2.5)),float(np.percentile(hq,97.5)))
###
Conf.int.95%: 6.783631842707721 7.382151120125547

```

Видно, что проверяемая медиана находится в пределах 95% доверительного интервала, поэтому нулевая гипотеза о равенстве медиан не отвергается.

## 6.5.Множественная проверка гипотез

Если провести  $m$  независимых тестов на уровне значимости  $\alpha=0.05$  , то: вероятность хотя бы одного ложного отклонения  $H_0$  (Family-Wise Error Rate, FWER):

$$FWER=1-(1-\alpha)^m$$

Например, при  $m = 20$  :

$$FWER=1-(1-0.05)^{20} \approx 0.64$$

Т.е. с вероятностью 64% после 20 тестов хотя бы один тест даст ложный результат! Обычно при проверке  $M$  верных гипотез из  $H_0$  с уровнем значимости  $\alpha$  по определению уровню значимости мы отвергаем примерно  $\alpha M$  верных гипотез.

Поэтому при проведении множественных статистических тестов увеличивается вероятность ложных открытий (False Positives, ошибок I рода) — ситуаций, когда мы ошибочно отвергаем верную нулевую гипотезу. Коррекция помогает контролировать общий

уровень ошибок.

Основные методы коррекции

- Поправка Бонферрони
- Метод Холма (Holm-Bonferroni)
- FDR (False Discovery Rate, Бенджамиини-Хохберг)

Как выбрать метод? Если нужен жёсткий контроль ошибок (FWER) - методы Бонферрони, Холма. Если нужен баланс между мощностью и ошибками (FDR) - Бенджамиини-Хохберг.

Рассмотрим детально эти методы. Пусть мы имеем выборки случайных величины одинаковой длины, взятые случайно из двух распределений с разными средними, и хотим выяснить к какому из распределений принадлежит каждая из этих выборок.

Код 41. Генерация тестового датасета и его множественная проверка одновыборочным критерием для среднего

```
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
N1=15000
N2=5000
L=20
data=np.concatenate([ss.norm(0,1).rvs((N1,L)),ss.norm(1,2).rvs((N2,L))])
labels=np.array([0]*N1+[1]*N2)
N=N1+N2
pva=[]
for i in range(N):
    x=data[i]
    m,s=x.mean(),x.std()
    T=(m-0)/(s/np.sqrt(L))
    pv=2*(1-ss.norm.cdf(abs(T)))
    pva.append(pv)
pva=np.array(pva)
pva_yp=(pva<0.05).astype(int)
pva_yp.shape,N,data.shape
print(confusion_matrix(labels,pva_yp))
#


|         | H0 pred | H1 pred |
|---------|---------|---------|
| H0 orig | 13983   | 1017    |
| H1 orig | 1881    | 3119    |


```

Из матрицы ошибок видно, что верно определяется примерно 93% основных гипотез, и примерно 62% альтернативных

### 6.5.1.Поправка Бонферрони

Поправка Бонферрони — это статистический метод, который строго контролирует вероятность хотя бы одной ложноположительной ошибки (Family-Wise Error Rate, FWER)

при проведении  $N$  множественных сравнений. Чтобы сохранить общий уровень значимости  $\alpha$ , нужно ужесточить критерий для каждого отдельного теста:

$$\alpha_{corr} = \frac{\alpha}{N}$$

или эквивалентно скорректировать p-value согласно условию:

$$pvalue_{corr,i} = \min(1, N pvalue_i)$$

Код 42. Корректировка результатов множественных тестов по методу Бонферони

```
pva_yp=(pva<0.05/N).astype(int)
pva_yp.shape,N,data.shape
print(confusion_matrix(labels,pva_yp))
#
```

	H0 pred	H1 pred
H0 orig	15000	0
H1 orig	4835	165

Из матрицы ошибок видно, что после поправки Бонферони верно определяется 100% основных гипотез, и примерно 3% альтернативных.

## 6.5.2.Метод Холма

Видно, что поправка Бонферони отбрасывает слишком много альтернативных гипотез. Метод Холма (Holm-Bonferroni) — это более гибкий способ контролировать FWER, сохраняя большую мощность. Он относится к нисходящим методам.

Это пошаговая процедура, которая:

- Ранжирует p-values по возрастанию:  $p_{v,i} \leq p_{v,i+1}$
- Сравнивает каждое  $p_{v,i}$  с адаптивным порогом начиная с  $i=1$ :

$$\alpha_i = \frac{\alpha}{N-i+1}$$

- Если  $p_{v,i} > \alpha_i$ , то все  $p_{v,j} \geq p_{v,i}$  считаются незначимыми

Эта процедура соответствует эффективному p-value:

$$p_{v,corr,i} = \min\left[1, \max\left((m-i+1)p_{v,i}, p_{v,corr,i-1}\right)\right]$$

Ключевое отличие от поправки Бонферони: порог становится мягче после каждого отклонённого  $H_0$ .

Код 43. Корректировка результатов проверки множественных гипотез по методу Холма

```
from statsmodels.stats.multitest import multipletests
```

```
reject, p_corrected, _, _ = multipletests(pva, alpha=0.05, method='holm')
```

```
print(confusion_matrix(labels,reject))
```

```
#
```

	H0 pred	H1 pred
H0 orig	15000	0
H1 orig	4834	166

Из матрицы ошибок видно, что после поправки Бонферни верно определяется 100% основных гипотез, и примерно 3% альтернативных. Метод Холма в данном случае не эффективнее поправки Бонферони.

### 6.5.3.Метод Бенджамини-Хохберга (FDR)

При массовой проверке гипотез традиционные методы (Бонферрони, Холм) становятся слишком строгими, отвергая все значимые результаты, соответствующие альтернативной гипотезе H1. FDR (False Discovery Rate) позволяет допустить некоторый процент ложных открытий, но сохранить больше реальных эффектов. Он относится к восходящим методам. Это пошаговая процедура, которая:

- Ранжирует p-values по возрастанию:

$$pv_i \leq pv_{i+1}$$

- Сравнивает каждое  $pv_i$  с адаптивным порогом начиная с  $i=N$ :

$$\alpha_i = \frac{i}{N} \alpha$$

- Если  $pv_i > \alpha_i$  то все  $pv_j \leq pv_i$  считаются значимыми.

Эта процедура соответствует эффективному p-value:

$$pv_{corr,i} = \min\left(p_i \frac{N}{i}, 1\right)$$

Ключевое отличие от поправки Бонферрони: порог становится мягче после каждого отклонённого  $H_0$ .

**Код 44. Корректировка результатов проверки множественных гипотез по методу Бенджамини-Хохберга**

```
reject, p_corrected, _, _ = multipletests(pva, alpha=0.05, method='fdr_bh')
print(confusion_matrix(labels,reject))
#
```

	H0 pred	H1 pred
H0 orig	14823	177
H1 orig	3455	1545

Из матрицы ошибок видно, что после поправки Бенджамини-Хохберга верно определяется 99% основных гипотез, и примерно 31% альтернативных. Метод Бенджамини-Хохберга в данном случае эффективнее и прямого тестирования (за счет более точного определения основных гипотез) и методов Бонферони и Холма (за счет более точного определения альтернативных гипотез).

## 7. Регрессия

### 7.1.Линейная регрессия

Чаще всего в машинном обучении с учителем решают два типа задач: регрессию и классификацию. Нулевое приближение регрессии - когда все прогнозируемые величины

постоянны и не зависят от входных параметров, нам почти никогда не интересно. Первое приближение - это линейная зависимость, когда мы находим линейную зависимость результата (целевой функции) от входных параметров. На основе этой зависимости мы делаем прогноз: если входные данные немного изменятся, новые точки лягут примерно на ту же линейную функцию.

Мы предполагаем, что целевая функция  $Y$  линейно зависит от аргументов плюс некоторое смещение  $b$  плюс шум  $\varepsilon$ .

$$Y = a_0X_0 + a_1X_1 + a_2X_2 + b + \varepsilon.$$

Это выражение описывает задачу линейной регрессии, где нужно найти коэффициенты  $a_i$ ,  $b$ . Можно расширить линейную регрессию, используя другие функции вместо  $X_i$ , и задача не усложнится, потому что неизвестные величины  $a_i, b$  входят в выражение линейно.

Чтобы найти оптимальные коэффициенты  $a_i$  и  $b$ , мы минимизируем специальный функционал. Этот функционал называется функцией потерь.

Простейшая функция потерь - это среднеквадратичное отклонение (MSE — Mean Squared Error). Функция потерь записывается как сумма квадратов разностей между реальным  $Y$  и модельным  $\hat{Y}$ . Ища коэффициенты  $a_i$  мы рассматриваем задачу минимизации квадратичной функции потерь.

Для наглядности обозначим экспериментальные значения как  $Y_{\text{эксп}}$ . Интегрирование (или суммирование) ведется по условному параметру  $T$ , который можно считать номером измерения в последовательности данных. Минимизация функции потерь записывается в виде:

$$\sum_T [Y_{\text{эксп}}(T) - (aX(T) + b)]^2 \rightarrow \min$$

Это означает, что мы ищем такие коэффициенты  $a$  и  $b$ , чтобы сумма квадратов разностей между экспериментальными и модельными значениями была минимальной.

Этот функционал является выпуклым - он имеет единственный глобальный минимум, где достигается нулевое значение (когда модель полностью совпадает с экспериментальными данными). В координатах  $(a, b)$  это выглядит как "чаша", в дне которой находится искомый минимум.

Благодаря выпуклости, для нахождения минимума можно использовать различные алгоритмы. Линейная регрессия популярна именно потому, что позволяет найти коэффициенты  $a$  и  $b$  аналитически (точным методом), а не только численно.

Для аналитического нахождения минимума приравниваем частные производные к нулю:

$$\begin{aligned} \sum [Y_{\text{эксп}} - (a \cdot X + b)] \cdot (-X) &= 0 \\ \sum [Y_{\text{эксп}} - (a \cdot X + b)] \cdot (-1) &= 0 \end{aligned}$$

Если у нас, например, 1 коэффициент  $a$  и 1 коэффициент  $b$ , мы получим систему из 2 уравнений (по числу неизвестных). После сокращения множителей уравнения примут вид:

$$\begin{aligned} \sum (Y_{\text{эксп}} X) &= \sum (aX^2 + bX) \\ \sum Y_{\text{эксп}} &= \sum (aX + b) \end{aligned}$$

Суммы  $X$  и их квадраты - это коэффициенты, которые можно вычислить. Если вынести их со знаком суммы получается система линейных уравнений на  $a, b$ . Эта система линейных уравнений позволяет точно найти оптимальные значения коэффициентов модели.

Систему линейных уравнений можно записать в матричной форме. Коэффициенты будут выглядеть так: суммы произведений, умноженные на столбец коэффициентов ( $a_0, a_1, \dots, b$ ). В общем виде получается система линейных уравнений, где есть суммы  $Y \cdot X$  и другие подобные выражения.

Такую систему можно решить, например, методом Крамера. Нужно вычислить определитель матрицы. Определитель — это "объем", ограниченный векторами системы. Сначала считаем определитель основной матрицы, затем заменяем столбцы на свободные члены и вычисляем вспомогательные определители. Каждый неизвестный коэффициент равен вспомогательному определителю (где столбец заменен на свободные члены), деленному на исходный определитель. Так можно напрямую вычислить все коэффициенты и решить систему.

#### Код 45. Определение коэффициентов линейной регрессии по методу Крамера

```
import numpy as np
def cramer_solve(x, y):
    n = len(x)
    sum_x, sum_y = sum(x), sum(y)
    sum_x2 = sum(xi**2 for xi in x)
    sum_xy = sum(xi*yi for xi, yi in zip(x, y))
    D0 = n * sum_x2 - sum_x**2
    a = (n * sum_xy - sum_x * sum_y) / D0
    b = (sum_x2 * sum_y - sum_x * sum_xy) / D0
    return a, b
x = np.random.rand(15)*10
y = 5*x+3+np.random.rand()*0.1
a,b=cramer_solve(list(x), list(y))
print(a,b)
#
5.00 3.04
```

Видно, что метод Крамера хорошо решает эту задачу.  
Попробуем решить эту задачу с помощью функций библиотеки scikit-learn.

#### Код 46. Определение коэффициентов линейной регрессии функциями библиотеки sklearn

```
from sklearn.linear_model import LinearRegression
def lr(x, y):
    x_reshaped = np.array(x).reshape(-1, 1)
    y_array = np.array(y)
    model = LinearRegression()
    model.fit(x_reshaped, y_array)
    a = model.coef_[0]
    b = model.intercept_
    return a, b
a,b=lr(list(x), list(y))
print(a,b)
#
5.000 3.048
```

Объект класса `LinearRegression` после обучения на данных возвращает коэффициенты: коэффициенты `a` (`coef_`) и смещение `b` (`intercept_`).

Важно отметить, что выбор функции потерь (критерия минимизации) - это сознательный выбор исследователя. Хотя чаще используют среднеквадратичную ошибку из-за возможности быстро и аналитически вычислить искомые коэффициенты, можно применять и другие выпуклые функции, имеющие один минимум. В сложных случаях приходится использовать численные методы оптимизации.

## 7.2.Метрики качества в задачах регрессии

При сравнении моделей и решении задач важную роль играют метрики качества. Метрики качества - это величины, которые показывают, насколько хорошее или плохое у вас решение. В отличие от функции потерь, которую в машинном обучении всегда нужно минимизировать, с метриками качества ситуация сложнее - некоторые нужно максимизировать, а другие минимизировать, в зависимости от их смысла.

Функция потерь сложна для интерпретации. Но если использовать простую функцию потерь, можно сказать: "У меня средний разброс значений вокруг прогноза составляет столько-то" (например, 1 день, 2 дня, 33 коровы и т. д.). Это можно интерпретировать.

Однако, если функционал сложный (например, с регуляризатором), функцию потерь уже нельзя интерпретировать. Чтобы оценить качество модели, используют метрики качества.

Основные метрики регрессии:

1. **MAE** (Mean Absolute Error) - средняя абсолютная разница между предсказанными и истинными значениями. Манхэттенское расстояние. Устойчива к выбросам. Наилучшее значение минимально (0).

$$MAE = \frac{1}{N} \sum_{i=1}^N |y_i - y_{model,i}|$$

Код 47. Оценка качества по метрике MAE

```
from sklearn.metrics import mean_absolute_error
x = np.random.rand(15)*10
y = x+np.random.rand()*0.5
print(mean_absolute_error(x,y))
#
0.177
```

2. **MSE** (Mean Squared Error) - среднее квадратов отклонений. Чаще используют RMSE (корень из MSE) - это аналог среднеквадратичного отклонения в статистике. Наилучшее значение минимально (0).

$$MSE = \frac{1}{N} \sum_{i=1}^N |y_i - y_{model,i}|^2$$
$$RMSE = \sqrt{MSE}$$

Код 48. Оценка качества по метрике MSE и RMSE

```
from sklearn.metrics import mean_squared_error,root_mean_squared_error
print(mean_squared_error(x,y))
print(root_mean_squared_error(x,y))
#
0.031 0.177
```

3. **R<sup>2</sup>** (коэффициент детерминации) - показывает, какую часть дисперсии данных объясняет модель. Наилучшее значение максимально (1).

$$R^2 = 1 - \frac{D(y - y_{model})}{D(y)}$$

- Идеальная модель: R<sup>2</sup>=1
- Наивный прогноз (константа = среднее): R<sup>2</sup>=0

- Плохая модель:  $R^2 < 0$  (хуже наивного прогноза)

#### Код 49. Оценка качества по метрике MSE и RMSE

```
from sklearn.metrics import r2_score
print(r2_score(x,y))
#
0.996
```

Эти метрики хорошо интерпретируются — по ним видно, насколько качественно модель описывает данные.

### 7.3. Переобучение и недообучение

Для линейной регрессии и классификации мы рассмотрели основные функционалы. Перейдем к проблеме переобучения. Рассмотрим пример: у нас есть данные, хорошо описываемые параболой ( $y = ax^2 + bx + c$ ). Мы настраиваем модель и получаем хорошее соответствие на имеющихся данных. Однако при поступлении новых точек оказывается, что истинная функция - синусоида, и наша параболическая модель даёт плохие предсказания на новых данных. Это и есть переобучение - модель слишком точно подстроилась под обучающие данные, потеряв обобщающую способность.

Возможна и обратная ситуация - недообучение, когда модель недостаточно сложна, чтобы уловить закономерности даже в обучающих данных (например, пытаемся аппроксимировать синусоиду прямой). Различают три набора данных: обучающий (для настройки модели), валидационный (для проверки) и тестовый (для окончательной оценки). Переобучение проявляется, когда ошибка на обучающих данных мала, а на валидационных - велика. Недообучение - когда ошибка на обучающем наборе выше, чем на валидационном или просто велика.

С недообучением борются увеличением сложности модели или числа итераций обучения. Переобучение - более серьёзная проблема. Существует много подходов избежать переобучения. Один из подходов - использование регуляризации, которую мы рассмотрим далее.

### 7.4. Мультиколлинеарность признаков и регуляризация

Эффект мультиколлинеарности, когда признаки  $X$  становятся линейно зависимыми — как минимум один из них становится равным приблизительно линейной комбинацией других векторов.

В контексте линейной регрессии, где мы решаем систему линейных уравнений вида  $X\beta = Y$ , мультиколлинеарность проявляется следующим образом: согласно методу Крамера, решение выражается через определители:  $\beta_i = \det(X_i)/\det(X)$ , где  $X_i$  - матрица с замененным  $i$ -м столбцом на вектор  $Y$ . Когда столбцы линейно зависимы, определитель матрицы  $X$  стремится к нулю. Хотя формально деление на ноль невозможно, на практике определитель становится чрезвычайно малым. Это приводит к тому, что коэффициенты  $\beta_i$  начинают резко возрастать, становясь очень большими по абсолютной величине. При этом некоторые коэффициенты принимают большие положительные значения, а другие - большие отрицательные, что является характерным признаком мультиколлинеарности.

#### Код 50. Оценка качества при мультиколлинеарных данных

```

import numpy as np
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_squared_error
n_samples = 100
true_coef = np.array([3.0, 2.0])
x1 = np.random.randn(n_samples)
x2 = x1+np.random.randn(n_samples) * 0.0001
y = true_coef[0] * x1 + true_coef[1] * x2+np.random.randn(n_samples) * 0.1
X = np.column_stack((x1, x2))
X_err = X + np.random.randn(n_samples,2)*1.
model = LinearRegression()
model.fit(X, y)
y_pred = model.predict(X)
mse_orig = mean_squared_error(y, y_pred)
y_pred = model.predict(X_err)
mse_noisy = mean_squared_error(y, y_pred)
print(mse_orig, mse_noisy, model.coef_)
#
0.0103 2386.31 [ 36.1237 -31.1314]

```

Видно, что при мультиколлинеарных данных линейная регрессия дает неверные коэффициенты, имеющие большие значения и противоположные знаки. При зашумленных тестовых данных дает очень большие ошибки, что говорит о неприменимости такой модели.

Существует несколько способов для борьбы с мультиколлинеарностью признаков:

- Удаление одного из коррелированных признаков
- Использование регуляризации
- Изменение системы координат (метод главных компонент)
- Увеличение размера выборки

Рассмотрим регуляризацию. Этот способ заключается в модификации функционала ошибок добавлением штрафного слагаемого.

В случае L2-регуляризации (Ridge, гребневой или тихоновской регуляризации) добавляется сумма квадратов коэффициентов с малым параметром  $\lambda$ :

$$\|X\beta - Y\|^2 + \lambda \|\beta\|^2 \rightarrow \min$$

где  $\lambda$  - параметр регуляризации, обычно принимающий небольшие значения ( доли или проценты от основного функционала). Ridge-регуляризация подавляет влияние выбросов, а не полностью их исключает.

#### Код 51. Оценка качества при регрессии по мультиколлинеарным данным (Ridge-регрессия)

```

from sklearn.linear_model import Ridge
model = Ridge(0.1)
model.fit(X, y)
y_pred = model.predict(X)
mse_orig = mean_squared_error(y, y_pred)
y_pred = model.predict(X_err)
mse_noisy = mean_squared_error(y, y_pred)
print(mse_orig, mse_noisy, model.coef_)
#
0.0103 9.9628 [2.4952 2.4948]

```

Видно, что при мультиколлинеарных данных Ridge регрессия дает более корректные коэффициенты, имеющие близкие значения. Таким образом часть проблем

мультиколлинеарности компенсируется. На зашумленных тестовых данных дает меньшие ошибки, чем простая линейная регрессия, что говорит о более высоком качестве такой модели.

Альтернативный подход - L1-регуляризация (Lasso), где добавляется сумма модулей коэффициентов:

$$\|X\beta - Y\|^2 + \lambda \|\beta\| \rightarrow \min$$

L1-регуляризация особенно эффективна для обнуления незначимых коэффициентов, поскольку при  $k=1$  она эквивалентна расчету медианы, которая устойчива к выбросам. Например, при наличии набора точек с одним выбросом, медианная регрессия проигнорирует этот выброс, в то время как метод наименьших квадратов сильно на него среагирует. Однако из-за недифференцируемости модуля в нуле, градиентные методы могут работать менее эффективно.

#### Код 52. Оценка качества при регрессии по мультиколлинеарным данным (Lasso-регрессия)

```
from sklearn.linear_model import Lasso
model = Lasso(0.1)
model.fit(X, y)
y_pred = model.predict(X)
mse_orig = mean_squared_error(y, y_pred)
y_pred = model.predict(X_err)
mse_noisy = mean_squared_error(y, y_pred)
print(mse_orig, mse_noisy, model.coef_)
#
0.0202 21.1449 [4.8936 0.]
```

Видно, что при мультиколлинеарных данных Lasso регрессия дает сравнительно корректные коэффициенты, некоторые из которых зануляются. Таким образом часть проблем мультиколлинеарности компенсируется. На зашумленных тестовых данных дает меньшие ошибки, чем простая линейная регрессия, что говорит о более высоком качестве такой модели.

Комбинированный подход Elastic Net объединяет оба вида регуляризации:

$$\|X\beta - Y\|^2 + \lambda (\alpha \|\beta\| + (1-\alpha) \|\beta\|^2) \rightarrow \min$$

где  $\alpha \in [0,1]$  - параметр, регулирующий баланс между L1 и L2 регуляризацией. При  $\alpha=0$  получаем Ridge-регуляризацию, при  $\alpha=1$  получаем Lasso регуляризацию.

#### Код 53. Оценка качества при регрессии по мультиколлинеарным данным (Elastic Net)

```
from sklearn.linear_model import ElasticNet
model = ElasticNet(alpha=0.1, l1_ratio=0.5)
model.fit(X, y)
y_pred = model.predict(X)
mse_orig = mean_squared_error(y, y_pred)
y_pred = model.predict(X_err)
mse_noisy = mean_squared_error(y, y_pred)
print(mse_orig, mse_noisy, model.coef_)
#
0.0391 9.2178 [2.4141 2.4098]
```

Видно, что при мультиколлинеарных данных Elastic Net регрессия дает сравнительно корректные коэффициенты, что-то среднее между Ridge и Lasso.

Также для регуляризации существует более общая метрика Минковского:

$$\left(\sum_i^N |\beta_i|^p\right)^{1/p}$$

где  $p \geq 0$ . При  $p=1$  получаем L1-норму, при  $p=2$  - L2-норму. На практике хорошие результаты часто достигаются при значениях  $p$  между 1 и 2.

Все эти методы регуляризации позволяют контролировать величину коэффициентов, предотвращая их неоправданный рост и снижая риск переобучения модели. По сути, регуляризационных подходов может быть множество - например, чтобы они включали сумму модулей коэффициентов в определенной степени. Чем меньше эта степень, тем устойчивее регуляризация к выбросам.

Особое внимание следует уделять выбору гиперпараметра регуляризации  $\lambda$ . Его подбор осуществляется после обучения модели: сначала фиксируется начальное значение (например, 0.01), затем анализируются полученные коэффициенты и точность. Если результаты неудовлетворительны, параметр варьируется с использованием валидационного набора данных и новой тренировки модели. Чаще всего перебор  $\lambda$  ведется через порядок.

## 7.5.Поиск гиперпараметров и кросс-валидация

При работе с небольшими наборами данных особое значение приобретают гиперпараметры - параметры, которые определяют либо функцию потерь, либо характеристики модели. Этих параметров может быть много, но их объединяет ограниченный набор возможных значений, которые нужно тоже перебирать. Включение гиперпараметров в процесс обучения создает сильную нелинейную зависимость, что может сильно усложнить поиск оптимальной модели. Поэтому обычно рассматривают разные обученные модели при разных значениях гиперпараметров, а затем выбирают оптимальный вариант на валидационном датасете.

Для оценки гиперпараметров используется метод кросс-валидации (перекрестной проверки). Стандартное разбиение на обучающую и валидационную выборки имеет недостаток - нет гарантии, что распределение данных в них одинаковое, особенно если в данных присутствуют временные тренды. Кросс-валидация решает эту проблему: исходный набор данных делят на  $K$  частей (например, 5), затем поочередно каждую часть используют для валидации, а остальные - для обучения. Это позволяет получить  $K$  значений функции потерь, по которым можно оценить среднее значение и разброс, проверив стабильность модели. Этот метод называется кросс-валидацией. Единственный её недостаток — скорость работы, так как модель обучается в  $K$  раз больше, что увеличивает время вычислений. Однако по сравнению с обычным разбиением на тестовую и обучающую выборки, кросс-валидация более надёжна.

Существует несколько способов выбора тестовых частей, которые сгруппированы в методах `model_selection`.

Основные методы разбиения:

1. **Случайное разбиение** (Train-Test Split) — простейшее разбиение на две части. Мы его уже рассматривали, поэтому подробно останавливаться не будем.
2. **KFold** — разбиение на  $K$  частей без перекрытия. Например, при  $K=10$  выборка делится на 10 фолдов без пересечений, и в каждом валидационном наборе используются разные индексы (0,1,...,9). В KFold валидационные выборки не перекрываются, что обеспечивает независимость оценок. Однако его недостаток в том, что пропорции классов могут не сохраняться. Например, если целевая переменная содержит 1000 нулей и 10 единиц, при случайному разбиении распределение единиц может оказаться неравномерным и модель обучится неверно.

Код 54. Тренировка модели методом кросс-валидации с разбиением по фолдам методом KFold

```
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score, KFold
import numpy as np
model = LinearRegression()
kf = KFold(n_splits=5, shuffle=True, random_state=42)
mse_scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error', cv=kf)
mse_scores = -mse_scores
print(f"Conf.int. MSE: {np.mean(mse_scores)} +/- {np.std(mse_scores)}")
#
Conf.int. MSE: 0.0113 +/- 0.0024
```

3. **StratifiedKFold** — аналогичен KFold, но сохраняет пропорции классов в тестовой и обучающей выборках, что важно при несбалансированных данных. StratifiedKFold сохраняет пропорции классов — он поддерживает их баланс. Например, если в исходных данных соотношение 1000 на 10, то и в тестовых и обучающих выборках оно останется примерно таким же.

4. Если нужно больше вариантов разбиения, чем позволяет KFold, используется **ShuffleSplit**. Он позволяет увеличить количество тестовых выборок за счёт перекрывающихся индексов. Например, в одном разбиении валидационная выборка может содержать индексы [0, 6], а в другом — [8, 6], то есть некоторые элементы повторяются. Это полезно при работе с небольшими датасетами, где нужно сгенерировать больше проверочных наборов.

5. Если важно сохранять пропорции классов, применяется **StratifiedShuffleSplit**, который обеспечивает сохранение пропорций между классами в валидационных и обучающих данных.

6. Ещё один метод — **Leave-One-Out**, где тестовая выборка состоит всего из одного элемента. Это используется для очень маленьких датасетов.

7. Если планируется прогноз данных во времени, валидационный датасет всегда должен идти за обучающим, чтобы имитировать процесс прогноза. Обычные методы разбиения при этом завышают качество прогноза. Для адекватного обучения в этом случае стоит использовать **TimeSeriesSplit**:

Код 55. Тренировка модели методом кросс-валидации с разбиением по фолдам методом TimeSeriesSplit

```
from sklearn.model_selection import TimeSeriesSplit
kf = TimeSeriesSplit(n_splits=5)
mse_scores = cross_val_score(model, X, y, scoring='neg_mean_squared_error', cv=kf)
mse_scores = -mse_scores
print(f"Conf.int. MSE: {np.mean(mse_scores)} +/- {np.std(mse_scores)}")
#
Conf.int. MSE: 0.0113 +/- 0.0046
```

Поскольку линейная регрессия — детерминированный метод, отличия в качестве моделей могут быть связаны с методом, использующимся для разбиения датасета.

## 7.6.Нелинейная регрессия

В случае когда у вас не все неизвестные параметры входят в модель линейно, и существует хотя бы один неизвестный параметр который входит задачу нелинейно, модель нелинейна. Чаще всего такие модели значительно сложнее решать, поэтому их стараются либо сводить к линейным (например за счет каких-то преобразований функций или переменных, входящих в модель), либо линеаризовать в малой окрестности значений искомых параметров.

В общем случае нелинейная регрессия - это всегда достаточно сложная задача и требует специальных методов для поиска коэффициентов. Это могут быть либо статистические методы, основанные на методе Монте-Карло и генетических алгоритмах, либо детерминированные методы, например метод градиентного спуска, метод Ньютона, поиск по сетке, ЕМ алгоритм. Либо комбинация этих методов.

В каких-то случаях нелинейную регрессию можно свести к линейной преобразованием функций или переменных.

Например, нелинейная модель

$$y_{mod} = A \cdot \sin(x+C) + D$$

сводится к линейной модели тригонометрическим преобразованием:

$$y_{mod} = E \cdot \sin(x) + F \cdot \cos(x) + D$$

Случай-же

$$y_{mod} = Ax^B$$

сводится логарифмированием к

$$\ln(y_{mod}) = A + B \cdot \ln(x)$$

Но в общем случае нелинейная регрессия - всегда проблема поиска минимума функционала невязки, решаемая специальными численными методами.

Один из способов оптимизация скорости поиска коэффициентов нелинейной регрессии - представить функцию в таком виде, чтобы максимально возможная часть неизвестных параметров входила линейно. По нелинейным вести поиск, по линейным - прямой расчет. Например, модель

$$y_{mod} = A \cdot \sin(W \cdot x + C) + D$$

преобразовываем к

$$y_{mod} = E \cdot \sin(W \cdot x) + F \cdot \cos(W \cdot x) + D$$

Перебираем по W по сетке или ищем его другим доступным нам методом, а коэффициенты E,F,D определяем на каждом шаге поиска аналитически, при заданном W.

Таким образом, решение задачи нелинейной регрессии всегда зависит от конкретной нелинейной модели, и иногда может быть сведено к линейной регрессии различными методами.

## 7.7.Линейная классификация.Логистическая регрессия

### 7.7.1.Логистическая регрессия

Перейдем к линейной классификации. Она возникает в задачах, где есть два набора точек, которые нужно разделить гиперплоскостью на два класса. Гиперплоскость - это поверхность размерностью на единицу меньше, чем само пространство. В 2D это линия, в 3D - плоскость, в 4D - трёхмерное подпространство и так далее. Уравнение любой гиперплоскости может быть выражено через скалярное произведение координаты пробной точки с вектором нормали  $\vec{w}$  плюс смещение b:

$$\langle \vec{w}, \vec{x} \rangle + b = 0$$

Мы решаем задачу так: если  $\langle \vec{w}, \vec{x} \rangle + b > 0$  - точка относится к нулевому классу , если  $\langle \vec{w}, \vec{x} \rangle + b < 0$  - ко первому. Это означает линейную разделимость.

Для поиска оптимальных коэффициентов удобно ввести величину M (функцию отступа, margin):

$$M = \sum_i y_i (\langle \vec{w}, \vec{x}_i \rangle + b)$$

Здесь  $y_i = -1$  метки для нулевого класса и  $y_i = 1$  - метки для первого класса. Это позволяет сформулировать задачу поиска гиперплоскости  $(\vec{w}, b)$  как максимизацию M.

Для максимизации функции отступа будем использовать простейший метод Монте-Карло (случайный поиск):

Код 56. Поиск оптимального разделения минимизацией функции отступа методом Монте-Карло

```
from sklearn.datasets import make_blobs
import numpy as np
def Yp(X,W,B): return (X @ W.T+B).sum(axis=-1)
def Mf(X,y,W,B): return (y*Yp(X,W,B)).sum()
X, y = make_blobs(n_samples=500, centers=2, n_features=2, random_state=42)
y=2*(y-0.5)
W, B, M = np.random.randn(2, 2), np.random.randn(1,2), -1e100
for _ in range(100):
    W_new = W + np.random.randn(*W.shape)*2.5
    B_new = B + np.random.randn(*B.shape)*2.5
    M_new = Mf(X,y,W_new,B_new)
    if M_new > M:
        W,B,M = W_new,B_new,M_new
print(f"Accuracy: {np.mean(y == np.sign(Yp(X,W,B)))}")
print("Parameters: ",W,B)
plt.scatter(X[:,0],X[:,1],c=np.sign(Yp(X,W,B)))
#
Accuracy: 1.0
Parameters:
[[-2.5972 3.3487] [ 4.4483 2.2295]]
[[-0.8204 4.5184]]
```

Видно, что обученный классификатор работает с точностью 100%.

Итоговый классификатор работает следующим образом: для исследуемой точки  $\vec{x}$  мы рассчитываем значение  $\langle \vec{w}, \vec{x} \rangle + b$  и определяем класс по знаку результата (больше нуля - класс +1, меньше нуля - класс -1)

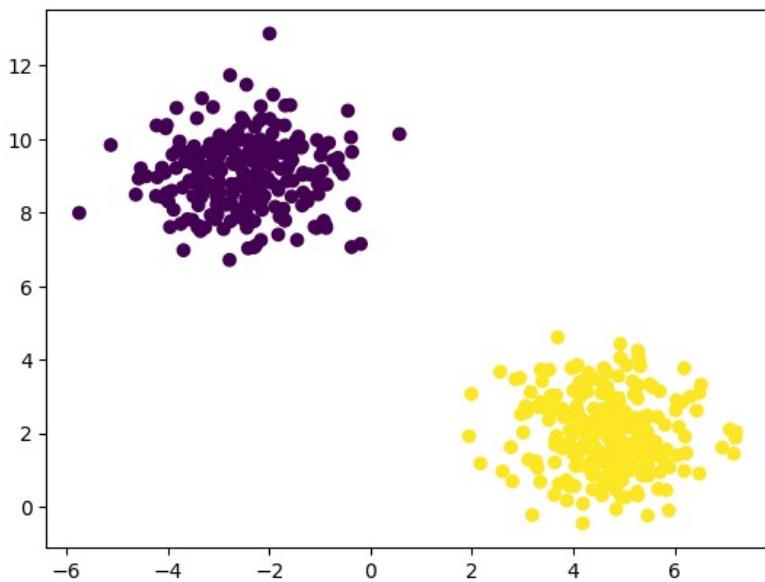


Рис.22. Результат разделения датасета линейным классификатором

Основной недостаток этого метода - негладкость функции отступа (она имеет ступенчатый характер), что затрудняет оптимизацию. Для использования более быстрых методов (например градиентного спуска) имеет смысл заменить эту функцию на более гладкий вариант, проходящий через максимумы исходной функции М. Например, можно использовать экспоненциальную функцию  $\exp(-M)$ , которая будет гладкой и дифференцируемой. Чаще всего используется логистическая функция потерь вида  $\log(1 + e^{(-M/T)})$ , где логарифм может быть от любого основания, а Т - произвольная величина («температура»).

Код 57. Поиск оптимального разделения минимизацией логистической функции потерь методом Монте-Карло

```
def Lf(X,y,W,B): return np.log(1+np.exp(-Mf(X,y,W,B)/X.shape[0]))
W, B, L = np.random.randn(2, 2), np.random.randn(1,2), 1e100
for _ in range(1000):
    W_new = W + np.random.randn(*W.shape)*2.5
    B_new = B + np.random.randn(*B.shape)*2.5
    Lf_new = Lf(X,y,W_new,B_new)
    if Lf_new < L:
        W,B,L = W_new,B_new,Lf_new
print(f"Accuracy: {np.mean(y == np.sign(Yp(X,W,B)))}")
#
Accuracy: 1.0
```

Этот подход почти всегда применяется в задачах классификации, включая нейронные сети. Логистическая функция потерь стала стандартным инструментом благодаря своим свойствам дифференцируемости и хорошей сходимости при градиентной оптимизации.

### 7.7.2.Метод опорных векторов (SVM)

Обобщением этого подхода является метод опорных векторов. Он строит

разделяющую поверхность, максимизируя расстояние между ближайшими точками разных классов до разделяющей гиперплоскости (отступ  $M$ ).

При обучении SVM находит параметры гиперплоскости: вектор нормали  $\vec{W}$  и смещение  $b$ . После обучения можно посмотреть предсказания — выше или ниже гиперплоскости лежит точка (в смысле положительная или отрицательная проекция вектора  $\vec{X}$  на нормаль  $\vec{W}$ ). Уравнение гиперплоскости имеет вид:

$$\vec{W} \cdot \vec{X} - b = 0$$

Алгоритм классификации имеет вид:

$$a(x) = \text{sign}(\vec{W} \cdot \vec{X} - b)$$

Максимизация отступа  $M$  проводится решением задачи:

$$\frac{1}{2} \|\vec{W}\|^2 + C \sum_{i=1}^l (1 - M_i(\vec{W}, b))_{\text{positive}} \rightarrow \min_{W, b}$$

$$M_{\text{positive}, i} = \langle \vec{W}, \vec{X}_i \rangle - b = \begin{cases} 1, & \text{if } Y_i = 1 \\ 0, & \text{if } Y_i = -1 \end{cases}$$

Часто вместо скалярного произведения используют функцию ядра, заменяя:  $\vec{W} \cdot \vec{X} \rightarrow K(W, X)$ . Ядра бывают разными, из нелинейных ядер чаще всего используется радиальная базисная функция (RBF):

$$K(\vec{X}_1, \vec{X}_2) = \exp(-\gamma \|\vec{X}_1 - \vec{X}_2\|^2)$$

Классификации подразделяют на:

1. Жесткие, когда постулируется 100% уверенность в классе ("это точно класс 1")
2. Мягкие, когда дается вероятностная оценка принадлежности к классу ("например 60% класс 1, 30% класс 2, 10% класс 3")

Мягкая классификация используется чаще, так как дает вероятностные оценки принадлежности к классам.

После обучения модели можно получить результаты в двух формах: жёсткая классификация (через возврат номера класса) и мягкая классификация (через возврат вероятностей каждого класса для этой точки). Они связаны между собой - из мягкой классификации всегда можно получить жёсткую, взяв класс с максимальной вероятностью, но обратное преобразование невозможно.

Сейчас большинство систем работают по принципу мягкой классификации, выдавая вероятности принадлежности к каждому классу.

Бинарная кросс-энтропия (и её многоклассовый вариант - кросс-энтропия) - одна из самых употребимых функций потерь. Она происходит из метода максимального правдоподобия и лежит в основе многих современных статистических методов:

$$BCE = - \sum_i (y_i \ln p_i + (1 - y_i) \ln (1 - p_i))$$

где  $p_i$  — выход модели (вероятность 1го класса), а  $y_i$  — учительская метка класса (0/1), суммирование ведется по элементам датасета.

### 7.7.3.Метрики качества в задачах классификации

Для анализа качества классификации в задачах часто строят матрицу ошибок. Если классификация на  $N$  классов, матрица выглядит так:

	Предсказанный			
Реальный	Класс 1	Класс 2	...	Класс N

Класс 1	$a_{1,1}$	$a_{1,2}$	...	$a_{1,N}$
Класс 2	$a_{2,1}$			
...	$a_{3,1}$			
Класс N	$a_{N,1}$			

В ячейках записывают, сколько раз реальный класс совпал с предсказанным. В идеале матрица должна быть диагональной (все недиагональные элементы — нули). Эта таблица эквивалентна обсуждавшейся ранее матрице сопряженности, используемой для сравнения номинальных величин.

#### Код 58. Пример построения матрицы ошибок

```
x=np.random.randint(0,5,1000)
y=x.copy()
np.random.shuffle(y[900:])
from sklearn.metrics import confusion_matrix
print(confusion_matrix(x,y))
#
[163 7 3 4 4]
[ 5 213 8 7 2]
[ 5 4 189 3 3]
[ 3 5 3 197 4]
[ 5 6 1 1 155]
```

Для бинарной классификации (классы P и N) матрица выглядит так:

	Предсказанный	
Реальный	Класс P	Класс N
Класс P	TP	FN
Класс N	FP	TN

Где:

- TP (True Positive) — правильно предсказанные позитивные случаи;
- TN (True Negative) — правильно предсказанные негативные случаи;
- FP (False Positive) — ошибка второго рода (негативный класс предсказан как позитивный);
- FN (False Negative) — ошибка первого рода (позитивный класс предсказан как негативный).

#### Код 58. Пример построения матрицы ошибок для задачи бинарной классификации

```
x=np.random.randint(0,2,1000)
y=x.copy()
np.random.shuffle(y[900:])
print(confusion_matrix(x,y))
#
[478 28]
[ 28 466]
```

Одна из наиболее простых метрик качества — ACCURACY (ACC):

$$ACCURACY = \frac{TP + TN}{TP + TN + FP + FN}$$

Метрика ACCURACY представляет собой отношение чисел на диагонали матрицы к общему количеству элементов и равна проценту верных угадываний.

#### Код 58. Пример вычисления метрики точности

```
from sklearn.metrics import accuracy_score
print(accuracy_score(x,y))
#
0.944
```

Существуют также метрики PRECISION и RECALL, которые похожи на ACCURACY, но рассчитываются иначе. В этих случаях берётся одна ячейка и делится либо на сумму по строке, либо на сумму по столбцу.

В некоторых задачах это удобнее. Например, если вам даны результаты и указано, какие классы считаются положительными, можно оценить, в каком количестве случаев была допущена ошибка. Precision показывает, сколько из предсказанных положительных случаев действительно верны.

$$PRECISION = \frac{TP}{TP + FP}$$

RECALL отражает, какая доля истинных положительных случаев была правильно определена.

$$RECALL = \frac{TP}{TP + FN}$$

Эти три метрики (ACCURACY, PRECISION, RECALL) хорошо работают, когда количество примеров в положительных и отрицательных классах примерно одинаково.

#### Код 59. Пример вычисления метрик PRECISION и RECALL

```
from sklearn.metrics import precision_score,recall_score
print(precision_score(x,y),recall_score(x,y))
#
0.9433 0.9433
```

Однако часто встречаются ситуации, когда классы сильно отличаются по количеству примеров (например, нулей значительно больше, чем единиц). В таких случаях ACCURACY, PRECISION, и RECALL плохо отражают реальное качество модели.

#### Код 60. Метрики качества при дисбалансе классов

```
x=np.random.randint(0,8,1000)
x[x>1]=0
y=x.copy()
np.random.shuffle(y[500:])
print(confusion_matrix(x,y))
print(accuracy_score(x,y))
print(precision_score(x,y),recall_score(x,y))
#
[822 52]
[ 52 74]
ACC: 0.896
```

```
PREC: 0.5873  
REC: 0.5873
```

Поэтому для задач классификации с дисбалансом классов используют другие метрики, например,  $F_1$ . Метрика  $F_1$  представляет собой гармоническое среднее PRECISION и RECALL:

$$F_1 = \frac{2 \text{PRECISION RECALL}}{\text{PRECISION} + \text{RECALL}}$$

Эта метрика эффективна при сильном дисбалансе классов.

```
Код 60. Метрика F1 при дисбалансе классов
```

```
from sklearn.metrics import f1_score  
print(f1_score(x,y))  
#  
0.5873
```

Вторая метрика - это ROC AUC (Area Under Curve, AUC ROC). Это площадь под кривой FP - TP (Receiver Operating Characteristic). Если построить график зависимости TP от FP для вашего датасета, то AUC ROC будет площадью под этой кривой. Чем больше эта площадь, тем лучше.

```
Код 60. Метрика AUC ROC при дисбалансе классов
```

```
from sklearn.metrics import roc_auc_score  
print(roc_auc_score(x,y))  
#  
0.7639
```

Аналогично существует метрика PR AUC (PRECISION-RECALL AUC, AUC PR) - тоже площадь, но под другой кривой - PRECISION-RECALL.

Три ключевые метрики ( $F_1$ , AUC ROC, AUC PR) устойчивы к дисбалансу классов и чаще всего стоит использовать их.

## 8.Анализ и прогноз квазипериодических рядов

До этого мы с вами в основном работали с неупорядоченными данными. У нас была какая-то куча данных, и мы пытались вылавливать корреляции, соотношения, и зависимости целевой функции  $u$  от входных векторов  $x$ . Подразумевалось, что данные упорядочены только по координатам вектора: координаты фиксированные, а сами данные могли идти в любом порядке, и результат не зависел от того, в каком порядке мы выстраивали эти данные.

Задачи с временными рядами сложнее. В данном случае считается, что у вас есть какая-то фиксированная переменная, которую мы назовём временем, вдоль которой все ваши данные упорядочены. Теперь у вас есть не куча, а ряд значений.

Будем считать, что функция  $u$  от  $t$  обладает следующими свойствами:

1. Она у вас скалярнозначная, то есть это действительные числа.
2. Она эквидистантная - это означает, что расстояние между отсчёты во времени у вас всегда сохраняется и равна  $\Delta t$ . То есть в данных нет пропусков или изменения временного масштаба.

На самом деле анализ временных рядов не ограничивается такими рядами, и чаще всего на практике у вас будут встречаться ряды, в которых есть пропуски, есть периоды в которых отсчёты получены не через одинаковые интервалы времени. Но пока мы это не учитываем. При возникновении такой ситуации, если у вас будет редкое число пропусков, то в традиционной постановке задачи ее решать уже нельзя. Необходимо будет либо выбирать только диапазоны, в которых выполняются условия выше, либо использовать более сложные методы анализа.

Детально можно посмотреть работы [5-7].

Итак, давайте сгенерируем тестовый временной ряд.

Код 60. Пример квазипериодической функции.

```
x=np.linspace(0,10,100)
y=np.sin(x*2)+x*0.3+np.random.randn(x.shape[0])*0.3
plt.plot(y)
```

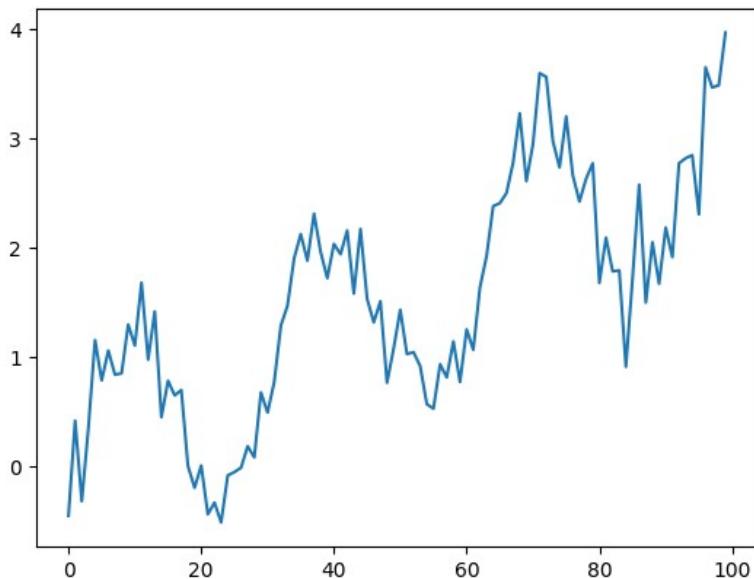


Рис.23. Пример квазипериодической функции.

Если мы построим нашу функцию на графике как зависимость от  $t$ , то увидим три особенности, которые нужно учитывать.

Первая особенность — тренд. Это плавное монотонное изменение скользящего среднего от функции, например ее возрастание.

Вторая характерная особенность — сезонность и циклы. Их можно объединить как периодические изменения. Сезонность — это когда изменения функции происходят с фиксированным периодом, а цикл — когда период может плавно меняться. И то, и другое говорит о квазипериодических вариациях функции.

Третий компонент — ошибка, которую нельзя предсказать. Она представляет собой шумоподобный ряд.

Код 61. Выделение элементов квазипериодической функции.

```
from statsmodels.tsa.seasonal import seasonal_decompose  
res=seasonal_decompose(y,model='additive',period=32)  
plt.plot(y,label='source')  
plt.plot(res.trend,'-',label='trend')  
plt.plot(res.season,'-.',label='season')  
plt.plot(res.resid,'--',label='noise')  
plt.legend()
```

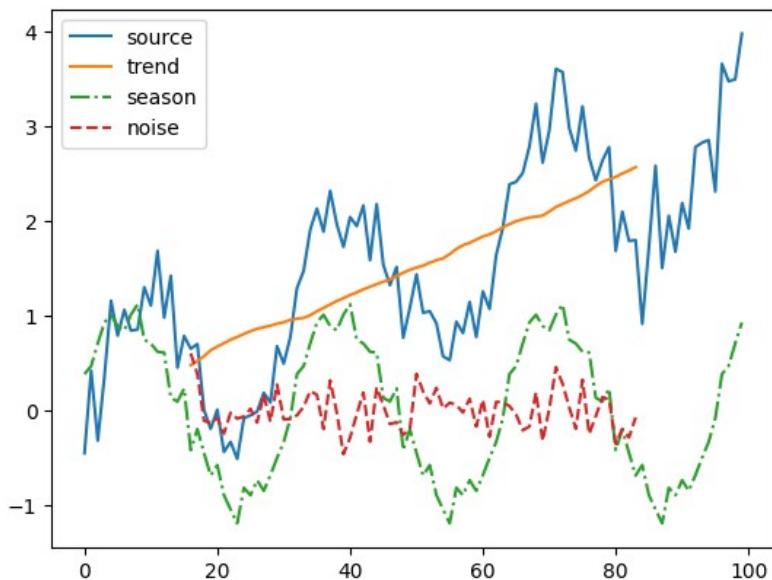


Рис.23. Элементы квазипериодической функции.

Проведенный анализ демонстрирует три компоненты:

1. Тренд — почти линейный, как мы и закладывали.
2. Периодическая составляющая — видно, что она имеет период 32 точки.
3. Остаток (шум) — варьируется случайно, его можно считать плохопредсказуемой ошибкой.

Квазипериодичность означает, что период может меняться. Амплитуда тоже может варьироваться. Например, звук синтезатора — это чистая гармоника, а звук скрипки

содержит обертона. На гитаре с эффектом дисторшн гармоники обрезаются, возникают резкие фронты. В акустическом канале появляются дополнительные колебания, которые могут быть разными в каждом цикле.

Таким образом, чистой периодичности в реальных данных почти не бывает. Обычно мы имеем дело с квазипериодическими процессами, где период или амплитуда плавно меняются. В экспериментальных данных обычно присутствует не один период, а несколько.

Стандартные задачи при работе с временным рядом — прогнозирование вперед (по времени) и фильтрация.

## 8.1.Фильтрация

Если задачей является нахождение  $A_i$  по заданным  $x_i, y_i$  – это задача регрессии. Если задачей является нахождение  $y_i$  по заданному  $x_i, A_i$  – это задача фильтрации.

При фильтрации часто выделяют два варианта фильтрации: фильтрацию с конечным импульсным откликом и фильтрацию с бесконечным импульсным откликом.

### 8.1.1.Фильтр с конечным импульсным откликом (КИХ-фильтр)

При фильтрации с конечным импульсным откликом используются коэффициенты зависимости выходного вектора  $y_{mod}$  от входного вектора  $x$ .

Выделение интересующей компоненты  $y$ , заданной уравнением регрессии:

$$y_{mod,i} = \sum_{j=0}^N A_j x_{i-j}$$

$A_i$  - импульсный отклик фильтра. Эти коэффициенты называются конечным импульсным откликом, потому что число коэффициентов, которые можно учесть обычно конечно.

Необходимо заметить, что если все коэффициенты  $A$  имеют одинаковый знак фильтрация будет выделять плавнomenяющуюся компоненту из  $x$  (по аналогии со средним), а если коэффициенты  $A$  меняют знак – то фильтрация будет выделять быстроменяющуюся компоненту из  $x$ . Таким образом, выбор коэффициентов  $A$  влияет на свойства фильтра.

Код 62. Фильтрация фильтром с конечным импульсным откликом

```
y=np.linspace(0,10,100)+np.random.randn(100)
yf=np.convolve(y,[0.25,0.25,0.25,0.25])
plt.plot(y,label='source')
plt.plot(yf,label='filtered')
plt.legend()
```

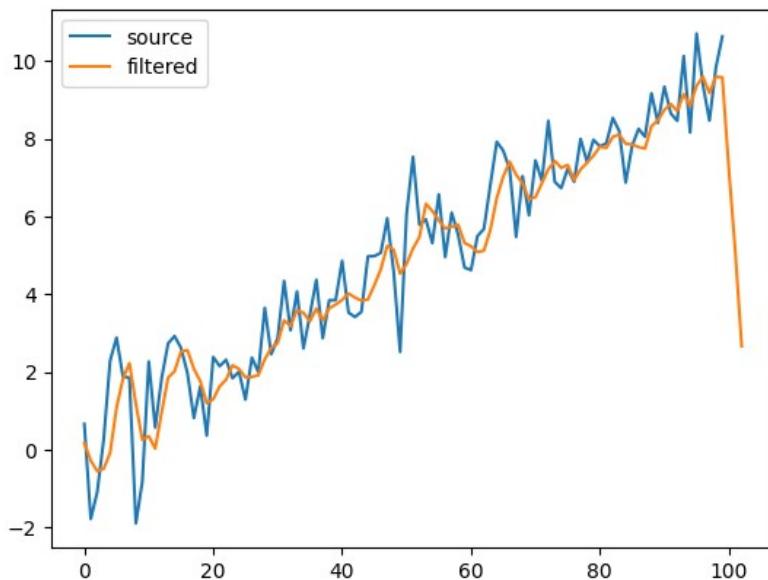


Рис.24. Фильтрация фильтром с конечным импульсным откликом

### 8.1.2.Фильтр с бесконечным импульсным откликом (БИХ-фильтр)

В случае если необходимо учесть зависимости не только короткие, которые определяются длиной импульсного отклика, но и длинные, то вам такая модель не подойдет. Поэтому используются фильтры с бесконечным импульсным откликом. В этом случае значение выходной функции  $y_{mod}$  зависит не только от входного вектора  $x$ , но и от значений выходной функции  $y_{mod}$  в предыдущие моменты времени.

Выделение интересующей компоненты  $y_{mod}$ , задается уравнением:

$$y_{mod,i} = \sum_{j=0}^N A_j x_{i-j} + \sum_{j=1}^M B_j y_{mod,i-j}$$

Такая модель позволяет сравнительно небольшим количеством коэффициентов обеспечить рекурсивную фильтрацию, и значительно удлиннить количество значений входного вектора  $x_i$ , влияющих на результат, обеспечив так называемый бесконечный импульсный отклик.

В частности в случае когда использование БИХ фильтра всего с двумя коэффициентами  $A_0$  и  $B_0$ , будет аналогично использованию КИХ фильтра с бесконечным ядром с экспоненциально убывающими коэффициентами.

Необходимо заметить что когда знаки регрессионных коэффициентов одинаковы, то это эквивалентно задаче взвешенного усреднения входного вектора  $x$ , и такая фильтрация выделяет в основном плавные изменения из входной функции. Если-же знаки регрессионных коэффициентов меняются, то такая фильтрация будет выделять из входного сигнала  $x$  скорее быстрые вариации, и ослаблять медленные. То есть подбором коэффициентов можно выделить те вариации, которые требуется дальнейшем анализировать.

Код 63. Фильтрация фильтром с бесконечным импульсным откликом
---

<pre>yf=np.zeros_like(y) A=0.7</pre>
--------------------------------------

```

for i in range(1,y.shape[0]):
    yf[i]=yf[i-1]*A+(1-A)*y[i]
plt.plot(y,label='source')
plt.plot(yf,label='filtered')
plt.legend()

```

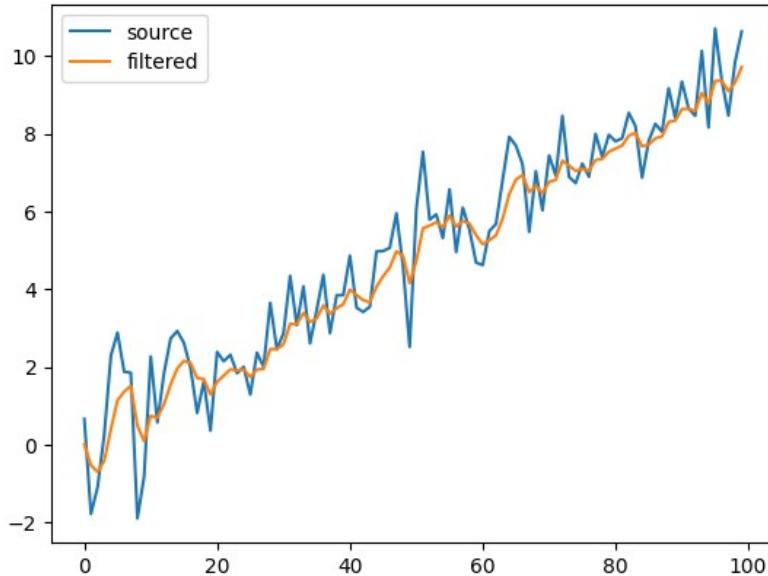


Рис.25. Фильтрация фильтром с конечным импульсным откликом

## 8.2.Авторегрессия

При прогнозировании вперед нам известен фрагмент ряда, и нужно предсказать его поведение в будущем или восстановить прошлое.

Простейшие методы прогнозирования вперед:

1. Наивный прогноз — предполагает, что будущее значение равно текущему (например, если сейчас температура  $-1^{\circ}$ , то через 15 минут будет  $-1^{\circ}$ ).
2. Линейный прогноз — учитывает скорость изменения (например, если час назад было  $+1^{\circ}$ , а сейчас  $-1^{\circ}$ , то через час, возможно, будет  $-3^{\circ}$ ).

Авторегрессионная модель один из способов такого прогноза. Она выражает текущее значение как сумму коэффициентов, умноженных на предыдущие значения.

-Если все коэффициенты, кроме первого, равны нулю, это наивный прогноз.

-Если коэффициенты подобраны иначе (например, 2 и -1), это линейный прогноз, учитывающий скорость изменения.

Одной из простейших моделей является **авторегрессия**. Авторегрессия используется, когда у вас нет ряда значений входной функции  $x$ , а есть просто ряд значений выходной функции  $y$ . В этом случае значение выходной функции предсказывается по ее значениям в предыдущие моменты времени. Эта задача тоже сводится к задаче регрессии.

Авторегрессия - зависимость значений ряда от истории его предыдущих значений (или функций от них), и обычно сводится к нахождению коэффициентов, обеспечивающих наилучшее (в смысле минимума функционала невязки) выполнение условия:

$$x_{mod,i} \approx A_1 \cdot x_{i-1} + A_2 \cdot x_{i-2} + A_3 \cdot x_{i-3} + \dots + A_N \cdot x_{i-N} + B$$

Также к задаче линейной регрессии относится и случай, когда прогнозирование выходной функции ведется не по линейной комбинации значений входной функции, а по линейной комбинации неких функций от этой входной функции.

$$x_{mod,i} \approx A_1 \cdot f_1(x_{i-1}) + A_2 \cdot f_2(x_{i-2}) + A_3 \cdot f_3(x_{i-3}) \dots + A_N \cdot f_N(x_{i-N}) + B$$

где  $f_i()$  - произвольные функции.

Неизвестные коэффициенты будут входить в это выражение линейно, и соответственно, можно достаточно легко посчитать значения этих коэффициентов.

Решение обоих случаев в рамках метода наименьших квадратов (т.е. при квадратичном функционале невязки) сводится к минимизации функционала вида:

$$\Omega = \sum (x_{mod,i} - x_i)^2 = min$$

то есть в конечном итоге также к решению системы линейных уравнений.

#### Код 64. Линейная авторегрессия

```
from sklearn.linear_model import LinearRegression
lr=LinearRegression()
y1=np.stack((y[:-3],y[1:-2],y[2:-1])).T
y2=y[3:]
lr.fit(y1[:-20],y2[:-20])
ym=lr.predict(y1)
print(lr.coef_,lr.intercept_)
plt.plot(y2,label='source')
plt.plot(ym,label='prediction')
plt.legend()
#
[0.13951739 0.17534272 0.57713449] 0.6069292709655594
```

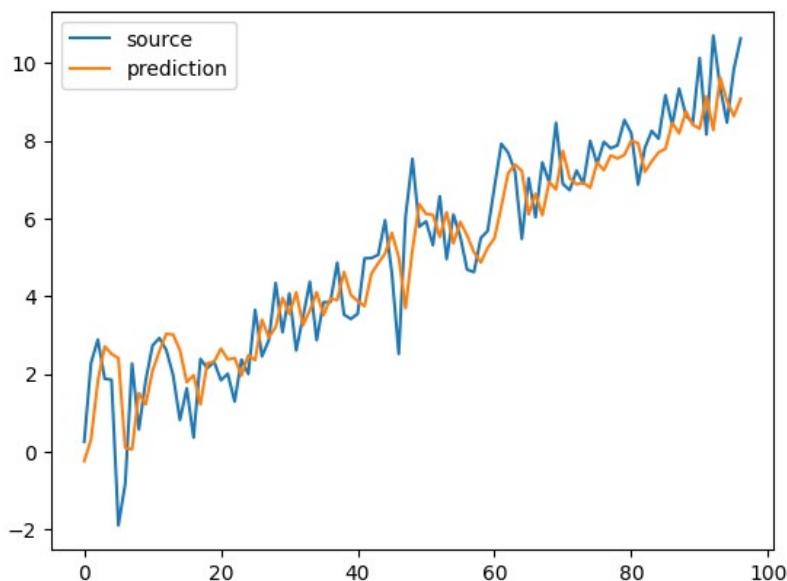


Рис.26. Линейная авторегрессия

Проверка качества авторегрессии может быть выполнена известными нами метриками качества — MAE, MSE,  $R^2$ :

#### Код 64. Метрики качества прогноза линейной авторегрессией

```
from sklearn.metrics import mean_squared_error,r2_score
print(mean_squared_error(y2,ym),r2_score(y2,ym))
#
1.2328 0.8440
```

### 8.3.Стационарность временных рядов

При вычислении регрессии важно проверить стационарность ряда - это позволяет использовать больше устойчивых алгоритмов. Стационарность бывает в широком и узком смысле.

В широком смысле ряд стационарен, если среднее и дисперсия в любом скользящем окне не меняются. Визуально можно оценить: если скользящее среднее в начале ряда одно, а в конце другое, но разница статистически незначима (проверяется например двухвыборочным Т-критерием), то ряд можно считать стационарным. Но если разница статистически значима - ряд нестационарен.

Стационарность в узком смысле более строгая: требует неизменности всех возможных корреляций между значениями ряда во времени. Стационарный в узком смысле ряд всегда стационарен в широком, но не наоборот. На практике обычно проверяют только широкую стационарность.

Для проверки наличия линейного тренда используют критерий Дики-Фуллера. Его нулевая гипотеза - ряд нестационарен (имеет линейный тренд). Критерий анализирует приращения между соседними значениями: если они не пропорциональны значениям ряда, то ряд стационарен. Есть модификации критерия для различных видов тренда.

#### Код 65. Проверка отсутствия линейного тренда в ряду критерием Дики-Фуллера

```
x=np.linspace(0,10,100)
y=np.sin(x*2)+x*0.3+np.random.randn(x.shape[0])*0.3
from statsmodels.tsa.stattools import adfuller
res=adfuller(y)
print('pvalue:',res[1])
#
pvalue: 0.5218
```

Если получили  $p\text{-value} < 0.05$  (при нулевой гипотезе о нестационарности) значит, ряд стационарен, а если  $p\text{-value} > 0.05$  — существенен линейный тренд. Видно, что критерий выявил в данных сильный линейный тренд.

Для преобразования нестационарного ряда в стационарный есть методы. Первый - метод Бокса-Кокса. Он аналогичен логарифмированию в акустике: человеческое ухо воспринимает звук в логарифмической шкале (децибелы - это  $20\lg(\text{амплитуда})$ ). Так, 10000 соответствует 80 дБ, а 10 - 20 дБ. Логарифмирование позволяет наглядно отображать данные с большим разбросом значений.

Один из методов преобразования нестационарного ряда в стационарный заключается в сжатии вариаций с помощью логарифмирования. Когда у нас сильные колебания, логарифм делает их более плавными - это вариант преобразования Бокса-Кокса.

Другой способ - возвести функцию в специальную степень. Например, извлечение корня: если были значения 1 и 100, после извлечения квадратного корня получаем 1 и 10 - вариации значительно уменьшаются. Это степенное преобразование Бокса-Кокса, где

параметр  $\lambda$  определяет тип преобразования:

- при  $\lambda=0$  выполняется логарифмирование
- при  $\lambda \neq 0$  - степенное преобразование

Метод эффективно работает, но имеет ограничение - он применим только к положительным рядам, так как логарифм от отрицательных чисел неопределен. Для знакопеременных рядов добавляют константу, делая все значения положительными (например, вычитая 1.1 минимального значения).

#### Код 66. Избавление от линейного тренда методом Бокса-Кокса

```
from scipy.stats import boxcox
yt = boxcox(y-y.min(),lmbda=0.05)
plt.plot(y,label='source')
plt.plot(yt,label='Box-Cox transform')
plt.legend()
res=adfuller(yt)
print('pvalue:',res[1])
#
pvalue: 4.7906e-13
```

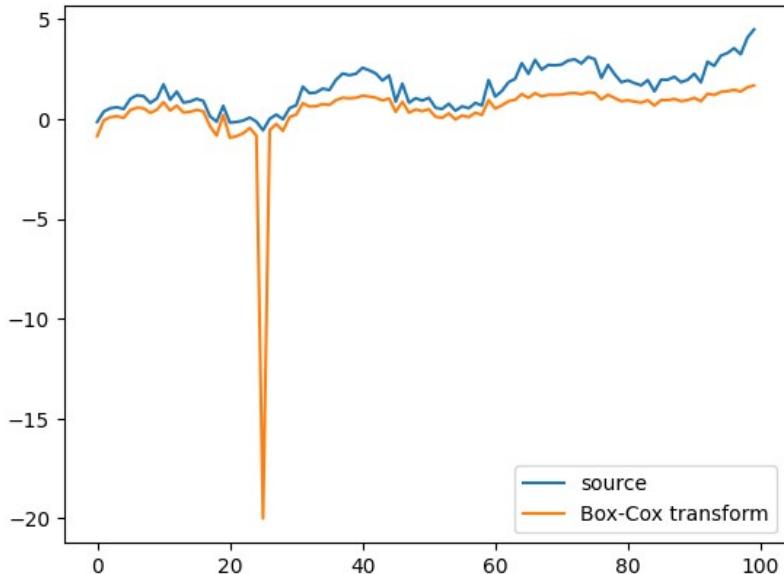


Рис 27. Избавление от линейного тренда методом Бокса-Кокса

То есть, хотя критерий преобразования Бокса-Кокса может изменить ряд, он не всегда делает его стационарным. У этого метода есть определенные диапазоны, где он не работает.

Один из побочных эффектов преобразования Бокса-Кокса — нормализация ряда. Это означает, что распределение квантилей становится близким к нормальному. Давайте проверим это с помощью QQ-plot.

Исходное распределение ненормально:

#### Код 67. Проверка нормальности исходного ряда методом квантиль-квантильного графика (QQ-plot)

```
import statsmodels.api as sm  
sm.qqplot(y, line='45')
```

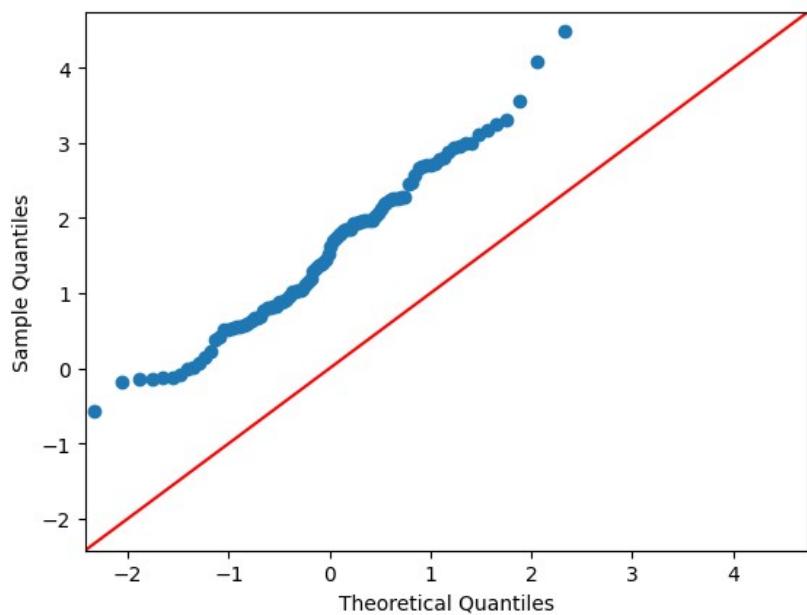


Рис 28. Проверка нормальности исходного ряда методом квантиль-квантильного графика (QQ-plot)

А получившееся — близко к нормальному за исключением выброса:

Код 68. Проверка нормальности ряда после преобразования Бокса-Кокса методом квантиль-квантильного графика (QQ-plot)

```
import statsmodels.api as sm  
sm.qqplot(yt, line='45')
```

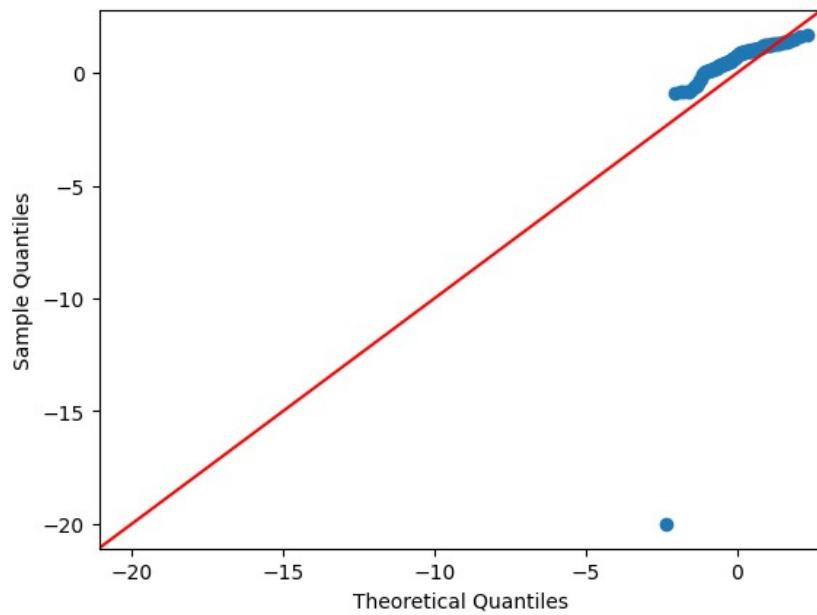


Рис 29. Проверка нормальности ряда после преобразования Бокса-Кокса методом квантиль-квантильного графика (QQ-plot)

Таким образом, преобразование Бокса-Кокса действительно нормализует ряд, делая его значения близкими к нормальным.

Более мощный и универсальный метод превращения нестационарного ряда в стационарный — дифференцирование. Суть метода в том, что из текущего значения вычитается предыдущее, формируя ряд из разностей. Если исходный ряд содержит тренд, то после дифференцирования его влияние значительно уменьшается.

#### Код 69. Избавление от линейного тренда методом дифференцирования

```

yt = y[1:]-y[:-1]
plt.plot(y,label='source')
plt.plot(yt,label='Diff.transform')
res=adfuller(yt)
plt.legend()
print('pvalue:',res[1])
#
pvalue: 0.1291

```

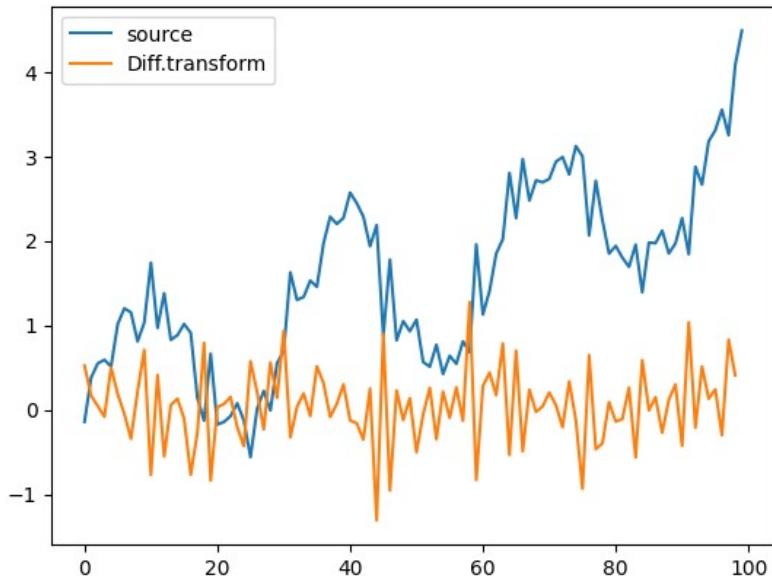


Рис 30. Избавление от линейного тренда методом дифференцирования

Если мы спрогнозируем дифференцированный ряд, то для восстановления исходного ряда нужно просуммировать все элементы. Это эквивалентно интегрированию. Таким образом, прогнозирование производного ряда почти не отличается от прогнозирования исходного, за исключением необходимости последующего интегрирования.

Еще один способ сделать ряд стационарным — сезонное дифференцирование. Оно позволяет избавиться от цикличности. Например, если есть медленный тренд и периодические колебания, вычитание значения в точке, отстоящей на длину периода, устранит циклическую компоненту.

#### Код 70. Избавление от линейного тренда методом сезонного дифференцирования

```
yt = y[32:]-y[:-32]
plt.plot(y,label='source')
plt.plot(yt,label='S.Diff.transform')
plt.legend()
res=adffuller(yt)
print('pvalue:',res[1])
#
pvalue: 0.0742
```

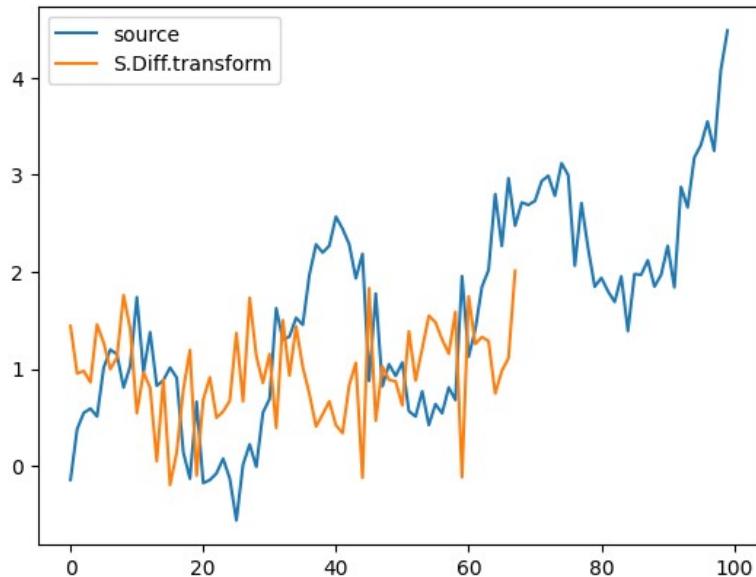


Рис 31. Избавление от линейного тренда методом сезонного дифференцирования

## 8.4. Время корреляции временных рядов

Чтобы проверить сколько элементов должно быть в авторегрессионной модели, нужно оценить связь текущих данных с предыдущими. Для этого можно использовать коэффициент линейной корреляции (коэффициент Пирсона).

Давайте начнём с построения графика зависимости текущих значений от предыдущих. Такая зависимость называется автокорреляционной функцией. Мы можем построить зависимость этого коэффициента от величины задержки  $\Delta t$ . При  $\Delta t=0$  коэффициент всегда равен 1 (полнная корреляция). Построив полную автокорреляционную функцию, мы видим, что коэффициент постепенно убывает, проявляя периодические колебания (из-за периодической составляющей в данных).

Код 71. Построение автокорреляционной функции ряда

```
from scipy.stats import pearsonr
ACF=[]
for i in range(90):
    if i==0: yc=y
    else: yc=y[:-i]
    ACF.append([pearsonr(y[i:],yc)[0],pearsonr(y[i:],yc)[1]])
ACF=np.array(ACF)
plt.plot(ACF[:,0])
```

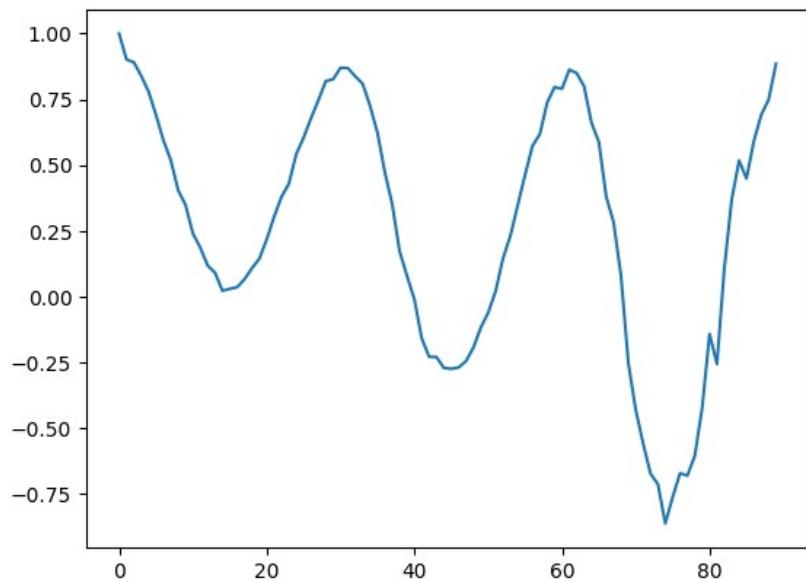


Рис 32. Автокорреляционная функция ряда

Момент, когда он достигает нуля, называется временем корреляции. Момент отсутствия корреляции можно проверить также по значению p-value:

Код 72. Поиск задержек, на которых корреляция отсутствует  
`plt.plot(ACF[:,1]>0.05)`

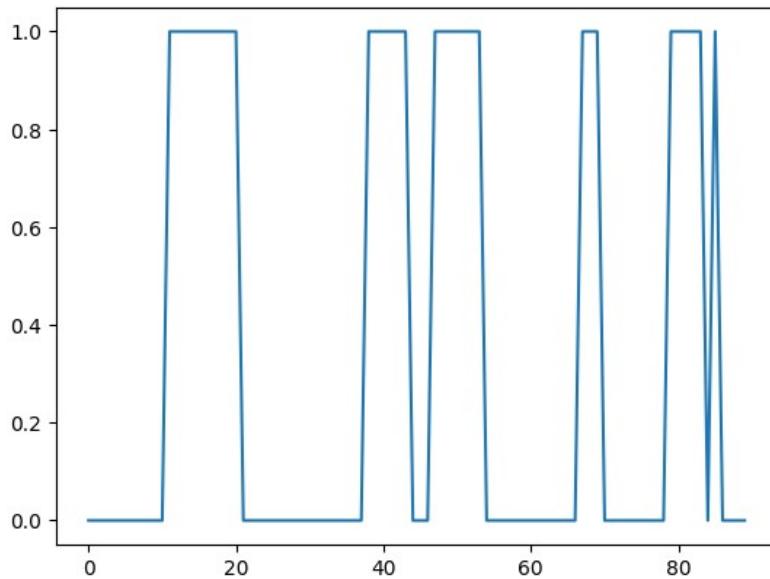


Рис 33. Поиск задержек, на которых корреляция отсутствует

Авторегрессионную модель имеет смысл строить только по предыдущим данным при  $\Delta t$ , где связь ещё значима.

## 8.5.Модели ARIMA и SARIMAX

Основу модели ARIMA составляет модель ARMA, составленная из двух частей. AR — это авторегрессионный прогноз, MA — прогноз скользящим средним.

$$y_t = A + \sum_{i=1}^p \phi_i y_{t-i} + \epsilon_t + \sum_{j=1}^q \theta_j \epsilon_{t-j}$$

Буква I в модели означает, что она может автоматически работать с дифференцированием/интегрированием ряда, и указывает на количество таких дифференцирований при применении модели. Как мы уже указывали, дифференцирование нужно для того, чтобы сделать ряд стационарным.

В модели ARIMA три параметра, и первый из них отвечает за порядок авторегрессии p, второй за количество дифференцирований, третий q — за порядок прогноза скользящим средним. Если мы установим p равным 1, а остальные параметры нулями - это будет означать, что модель зависит только от предыдущего значения ряда.

Код 73. Прогноз ряда моделью AR(14)

```
from statsmodels.tsa.arima.model import ARIMA  
model = ARIMA(y[:-10], order=(14,0,0))  
model_fit=model.fit()  
yp = model_fit.predict(start=0, end=y.shape[0])  
plt.plot(y,label='source')  
plt.plot(yp,label='prediction')  
plt.plot([y.shape[0]-10,y.shape[0]-10],[0,5])
```

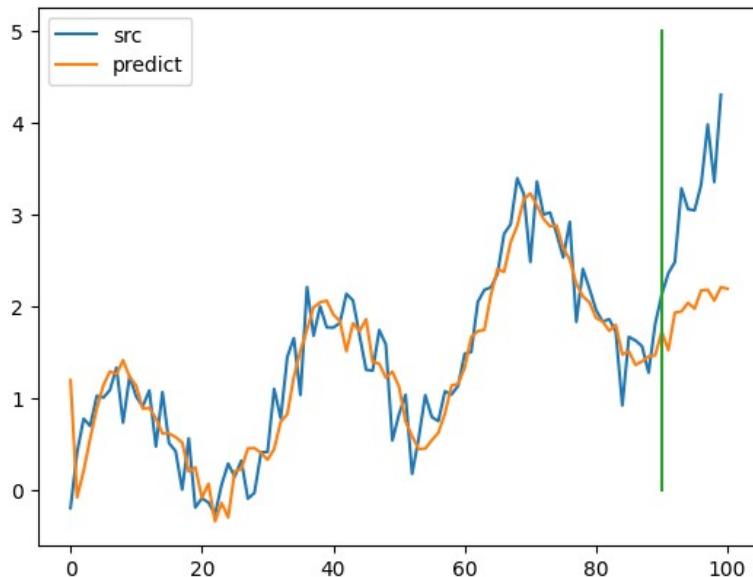


Рис.34. Прогноз моделью AR(14). Вертикальная линия отделяет обучающий датасет от валидационного

Посмотрим, как повлияет использование стационарного ряда

Код 74. Прогноз ряда моделью ARI(14,1)

```

from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(y[:-10], order=(14,1,0))
model_fit=model.fit()
yp = model_fit.predict(start=0, end=y.shape[0])
plt.plot(y,label='src')
plt.plot(yp,label='predict')
plt.plot([y.shape[0]-10,y.shape[0]-10],[0,5])
plt.legend()

```

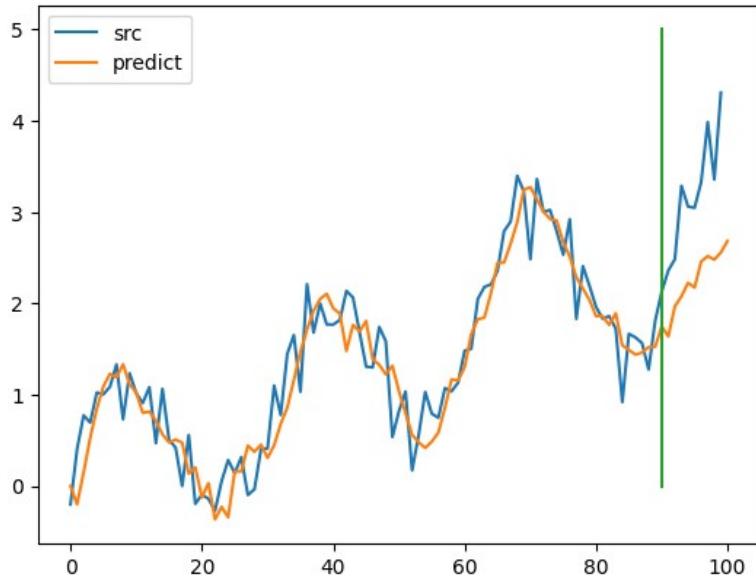


Рис.35. Прогноз моделью ARI(14,1). Вертикальная линия отделяет обучающий датасет от валидационного

Теперь рассмотрим другой пример. Представим, что у нас есть последовательность, сгенерированная случайным генератором чисел. Если попытаться спрогнозировать следующее значение на основе предыдущих с помощью авторегрессии, что получится?

Поскольку случайные числа независимы, модель не сможет выявить закономерностей и будет предсказывать просто среднее значение ряда. Если среднее равно нулю, прогноз всегда будет нулевым.

Теперь усложним задачу. Возьмём случайный ряд с нулевым средним и применим к нему свёртку с ядром из пяти элементов. Это означает, что каждое новое значение ряда будет вычисляться как взвешенная сумма пяти предыдущих случайных чисел. Такой ряд уже не будет полностью случайным — его значения окажутся коррелированными, поскольку соседние точки зависят от перекрывающихся наборов исходных чисел. Это означает, что авторегрессия может дать осмысленный прогноз.

Попробуем обучить модель на таком "фильтрованном шуме". Этот подход называется моделью скользящего среднего (MA) и основан на идее, что любую функцию можно представить как авторегрессию от некоторой случайной величины. То есть можно подобрать такой случайный ряд, из которого исходная функция будет восстанавливаться через авторегрессионную модель. Учет всех трех методов — авторегрессия (AR), дифференцирование/интегрирование(I) и прогноз скользящего среднего (MA) формирует модель ARIMA.

### Код 75. Прогноз ряда моделью ARIMA(14,1,10)

```
from statsmodels.tsa.arima.model import ARIMA
model = ARIMA(y[:-10], order=(14,1,10))
model_fit=model.fit()
yp = model_fit.predict(start=0, end=y.shape[0])
plt.plot(y,label='src')
plt.plot(yp,label='predict')
plt.plot([y.shape[0]-10,y.shape[0]-10],[0,5])
plt.legend()
```

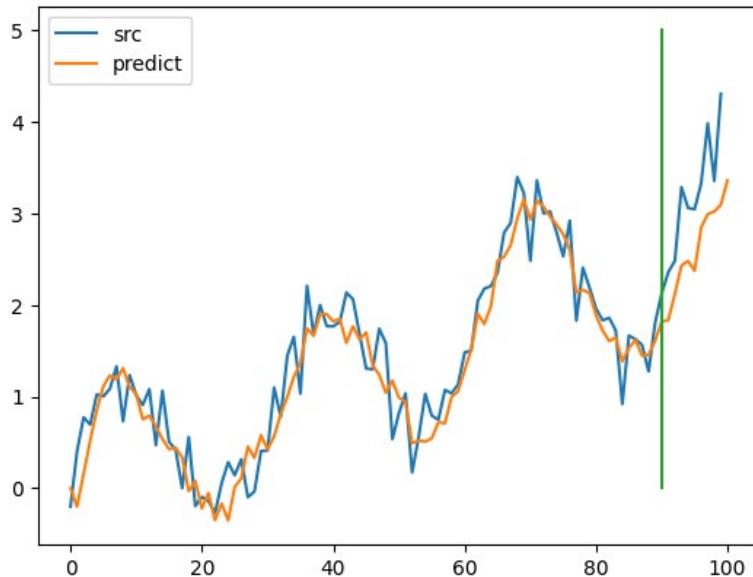


Рис.36. Прогноз моделью ARIMA(14,1,10). Вертикальная линия отделяет обучающий датасет от валидационного

Видно, что прогноз существенно улучшился.

Теоретически, для любого ряда можно подобрать параметры  $p$  и  $q$  так, чтобы модель ARIMA описывала его с любой точностью. Однако слишком большие  $p$  и  $q$  приводят к вычислительной сложности и риску переобучения, поэтому на практике используют компактные модели.

То есть чем больше у модели свободных параметров, тем точнее она у вас пропишет датасет, и тем выше ее способность к переобучению. То есть если  $p+q$  достаточно большое ( $p+q$  больше чем число элементов ряда) - вы пропишете этот ряд с идеальной точностью. Чем больше  $p+q$  - тем выше точность аппроксимации. Но аппроксимация и прогноз — разные вещи, если функция хорошо аппроксимирована моделью, это еще не значит что модель будет хорошо предсказывать новые данные.

Следующий уровень качества модели — учет сезонности. Итак, предположим у нас есть жесткая периодика в данных. Мы посмотрели на данные и сказали: "О, у нас каждые 32 точки появляется максимум". Как это учесть? Для этого используется модель SARIMA — сезонная ARIMA.

Модель имеет вид:

$$y_t = A + \sum_{i=1}^p \phi_i y_{t-i} + \epsilon_t + \sum_{j=1}^q \theta_j \epsilon_{t-j} + \sum_{i=1}^P \phi_i y_{t-ki} + \sum_{j=1}^Q \theta_j \epsilon_{t-kj}$$

и включает в себя сезонные регрессии, сезонные дифференцирования и сезонные скользящие средние. Сезонный — значит взятый через какой-то промежуток времени (сезон S).

Мы добавляем в модель ARIMA дополнительные зависимости от величин, отстоящих на период, кратный нашему периоду. Например, если мы измеряем температуру каждые 15 минут, мы берём первые 10 членов (2.5 часа), а зависимость от 20-летнего периода учитываем через температуру 20 лет назад, 40 лет назад и так далее.

То же самое делаем со случайной величиной в MA-части: строим регрессию с периодом в 20 лет. Это позволяет уменьшить количество коэффициентов и оптимизировать быстродействие. Такая модель называется SARIMA - у неё остаются параметры p,q от ARIMA, плюс добавляются параметры P,Q для периодической составляющей, и параметр s (в нашем случае 20 лет) - длина периода. Хотя обычно берут самый короткий период, а мы здесь взяли 20 лет.

Подбор параметров — это отдельная сложная задача, так как их много.

В библиотеке statsmodels реализовано через функцию SARIMAX с 7 параметрами:

- p,d,q - из ARIMA;

- P,D,Q,s - сезонные компоненты и длительность сезона.

Проверим, как это работает, добавив всего одно сезонное дифференцирование с сезоном 32 точки, известном нам ранее:

Код 76. Прогноз ряда моделью SARIMA(14,1,1,0,1,0) с сезоном 32

```
from statsmodels.tsa.statespace.sarimax import SARIMAX
model = SARIMAX(y[:-10], order=(14,1,1), seasonal_order=(0,1,0,32))
model_fit=model.fit()
yp = model_fit.predict(start=0, end=y.shape[0])
plt.plot(y,label='src')
plt.plot(yp,label='predict')
plt.plot([y.shape[0]-10,y.shape[0]-10],[0,5])
plt.legend()
```

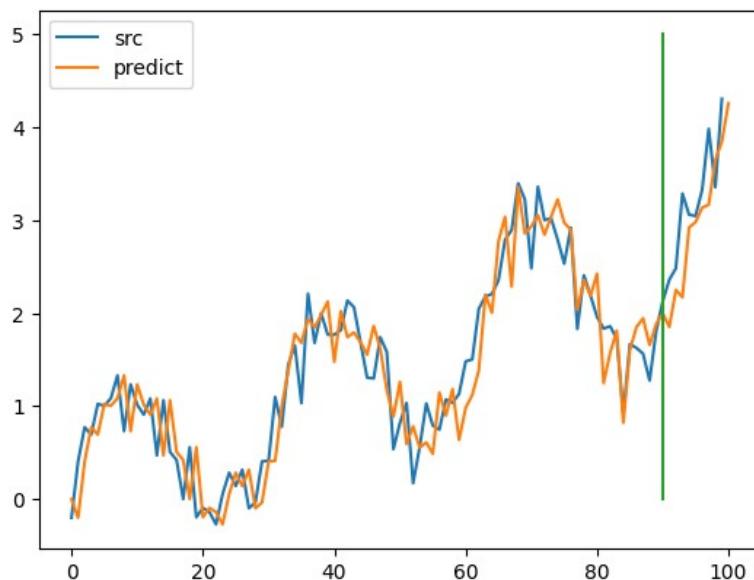


Рис.37. Прогноз ряда моделью SARIMA(14,1,1,0,1,0) с сезоном 32. Вертикальная линия отделяет обучающий датасет от валидационного

Для проверки качества модель настраивается на обучающей выборке, а проверяется на валидационной.

Подбор параметров делается перебором, чаще всего минимизируя критерий AIC или BIC. Критерии AIC и BIC учитывают логарифм правдоподобия и штрафует за избыток параметров.

Создание полной модели SARIMA, а тем более подстройка ее гиперпараметров может занять очень много времени, так как модель сложная и требует долгих вычислений и больших объемов памяти. Поэтому модель чаще всего используется при небольшом наборе данных и только одной выделенной периодичности. Для больших периодов нужны более сложные модели.

Для ускорения поиска гиперпараметров некоторые параметры лучше заранее зафиксировать. Один из способов — использовать автокорреляционную функцию (АКФ). Если АКФ в какой-то точке обращается в ноль (например, при  $lag=0$  или другом значении), это означает, что дальнейшие значения уже не коррелируют с предыдущими. По этому признаку можно определить необходимую длину авторегрессии (параметр  $p$  или  $P$ ). При наличии сезонного хода, рекомендуется все параметры ARMA брать меньшими длительности сезона.

Ещё один полезный инструмент — частичная автокорреляционная функция (ЧАКФ). Она строится на остатках после авторегрессии:

1. Сначала строится авторегрессия порядка  $N$

2. Затем вычисляется корреляция остатков — разниц между регрессией и реальными значениями. Это и будет точка частичной корреляционной функции на задержке  $N$ .

ЧАКФ помогает определить параметр  $q$  (для скользящего среднего, MA) или  $Q$  (для сезонного скользящего среднего, MA), так как показывает, на каком  $lag$  остатки перестают коррелировать.

Как строить АКФ и ЧАКФ? Можно использовать библиотеку statsmodels в Python:

- `plot_acf()` — строит автокорреляционную функцию.
- `plot_pacf()` — строит частичную автокорреляционную функцию.

Интерпретация графиков.

- Если АКФ медленно затухает, это значит, что ряд зависит от большого числа предыдущих значений (параметр  $p$  должен быть большим).
- Если в АКФ есть выраженные пики на кратных расстояниях (например, 12, 24, 36), это указывает на сезонность.

Таким образом, АКФ и ЧАКФ помогают сократить перебор параметров, задав разумные начальные значения.

После построения модели необходимо проверить ее полноту — что мы все учили в модели и не требуется ее усложнение. Для этого обычно анализируют остатки (разность между рядом значений и его прогнозом) на предмет стационарности, равенства нулю среднего и отсутствию корреляции:

- 1) Они должны быть центрированы около нуля (гистограмма)
- 2) Критерий Дики-Фуллера проверяет стационарность ( $p\text{-value} < 0.05$ )
- 3) Критерий Лонга-Бокса проверяет отсутствие автокорреляции

## 8.6.Разложение по ортогональным системам функций.Спектры

Еще одним важным методом анализа рядов являются матричные преобразования. Матричных преобразований мы встречали много, сейчас рассмотрим еще один тип матричных преобразований - разложение по ортогональным наборам функций. Такие преобразования называются спектральными представлениями. Детально можно посмотреть

работу [7].

Рассмотрим исходное изображение

Код 77. Просмотр тестового изображения

```
import matplotlib.pyplot as plt
import matplotlib.image as mpimg
img = mpimg.imread('TEST.png')
imgplot = plt.imshow(img,cmap='gray')
```

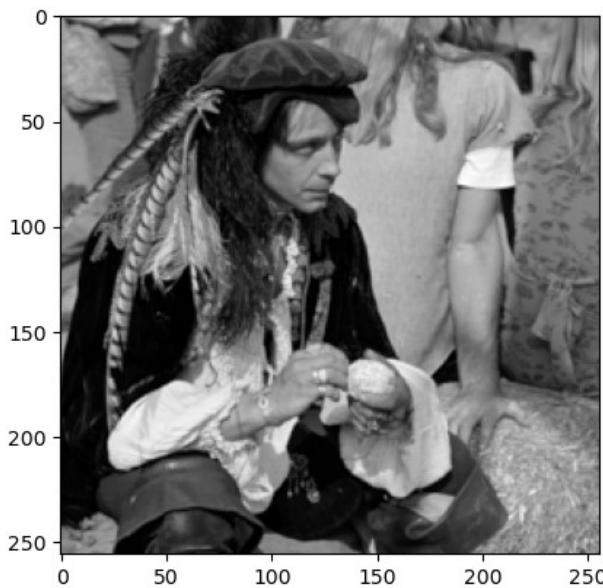


Рис. 38. Пират. Изображение 5.3.01 из <https://sipi.usc.edu/database/database.php?volume=misc>

### 8.6.1.Преобразование Хаара

Изображение можно разделить на плавнomenяющуюся часть и резкоменяющуюся. Для этого достаточно к двум соседним точкам применить разложение на полусумму и полуразность:

$$\begin{pmatrix} 1/2 & 1/2 \\ -1/2 & 1/2 \end{pmatrix} \begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} \frac{y+x}{2} \\ \frac{y-x}{2} \end{pmatrix} = \begin{pmatrix} x_{1,new} \\ x_{2,new} \end{pmatrix}$$

В более общем виде, сохраняющем длину новых векторов, это двухточечное преобразование Хаара.

$$H \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_{1,new} \\ x_{2,new} \end{pmatrix}$$
$$H = \begin{pmatrix} 1/\sqrt{2} & 1/\sqrt{2} \\ -1/\sqrt{2} & 1/\sqrt{2} \end{pmatrix}$$

здесь  $H$  — с точностью до множителя матрица Хаара.

Фактически для двух чисел оно преобразует два числа в их полусумму и полуразность. Если два числа - соседи на изображении, то полусумма будет выделять

плавные изменения, а полуразность - резкие. Из получившихся точек можно построить два изображения, соответствующих плавным цветам и резким границам:

#### Код 78. Преобразование Хаара

```
import numpy as np
from math import sqrt
gs_img = img[:, :]
gs_img1D=np.reshape(gs_img,gs_img.shape[0]*gs_img.shape[1])
gs_img1D_padded=gs_img1D[:-1]
slow=np.convolve(gs_img1D_padded,[1/sqrt(2),1/sqrt(2)])
fast=np.convolve(gs_img1D_padded,[1/sqrt(2),-1/sqrt(2)])
fig,axs=plt.subplots(1,2,figsize=(10,5))
axs[0].imshow(slow.reshape(256,256),cmap='gray')
axs[1].imshow(fast.reshape(256,256),cmap='gray')
```

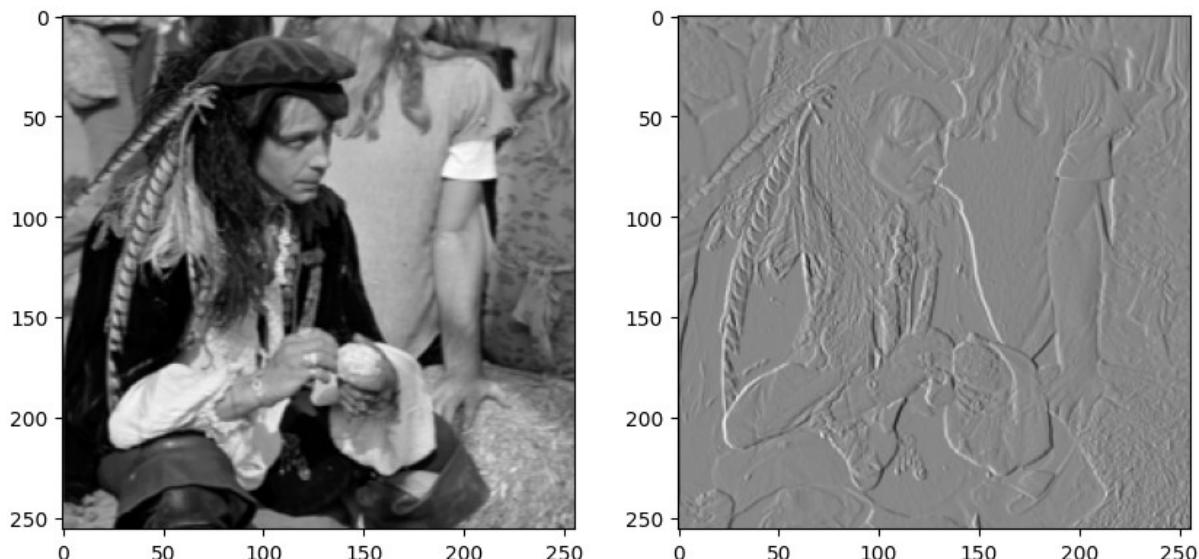


Рис. 39. Двухточечное преобразование Хаара. Слева — плавная часть изображения, справа - быстременяющаяся.

Преобразование Хаара обладает обратным преобразованием:

$$H^{-1} H \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$$

причем обратная матрица к  $H$  равна ее транспонированному значению:

$$H^{-1} = H^T$$

В более общем случае  $k=2^N$  точек преобразование Хаара имеет вид :

$$H_{k=2^N, m, n} = \begin{cases} \sqrt{(2^m)}, & n \in (\frac{k-1}{2^m}, \frac{k-1/2}{2^m}) \\ -\sqrt{(2^m)}, & n \in (\frac{k-1/2}{2^m}, \frac{k}{2^m}) \\ 0, & \text{other cases} \end{cases}$$

## 8.6.2. Преобразование Уолша

Другим матричным преобразованием является преобразование Уолша. Оно также способно преобразовывать последовательности из  $2^N$  точек в  $2^N$  коэффициентов. Это эквивалентно разложению последовательности в ряд по базисным функциям Уолша  $W_n(t)$ :

$$x(t) = \sum_{n=1}^{2^N} A_n W_n(t)$$

Формула очень напоминает регрессию.

Получившиеся коэффициенты  $A_n$  показывают, с какой амплитудой в нашем ряду участвуют различные функции  $W_n(t)$ . Набор коэффициентов  $A_n$  называется спектром.

Формулу можно переписать в матричном виде:

$$\begin{pmatrix} x_1 \\ \dots \\ x_{2^N} \end{pmatrix} = H_{2^N} \begin{pmatrix} A_1 \\ \dots \\ A_{2^N} \end{pmatrix}$$

где  $H_{2^N}$  — матрица размера  $2^N \times 2^N$ , называющаяся матрицей Адамара порядка  $2^N$  и состоящее из  $2^N$  функций Уолша.

Матрица определяется рекурсивно через матрицы Адамара предыдущего порядка и матрицу  $H_1 = (1)$ :

$$H_{2^N} = \begin{pmatrix} H_{2^{N-1}} & H_{2^{N-1}} \\ H_{2^{N-1}} & -H_{2^{N-1}} \end{pmatrix}$$

Развернем матрицу пикселов изображения в ряд, и получившийся ряд значений разложим в спектр Уолша.

Код 79. Преобразование Уолша

```
from sympy import fwht
spec=fwht(gs_img1D)
plt.plot(spec)
```

Получившийся спектр состоит из пиков:

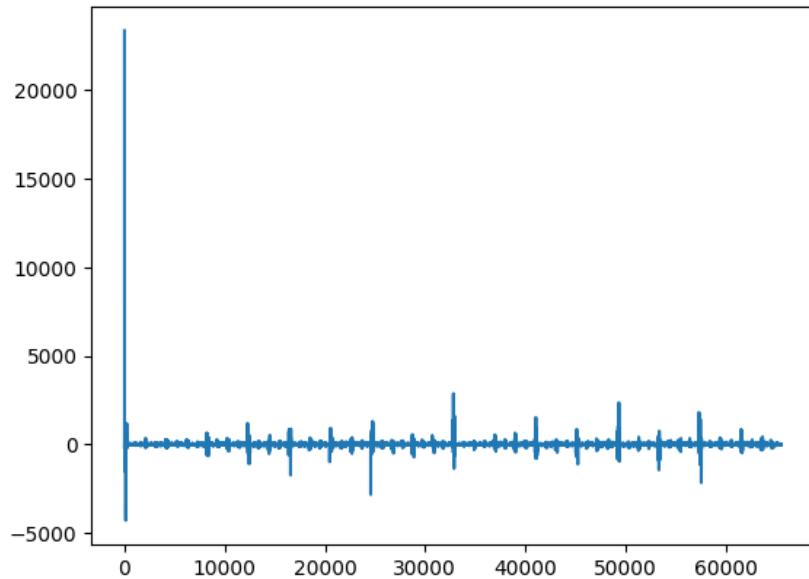


Рис.40. Спектр Уолша от тестового изображения

Преобразование Уолша обратимо, что значит существует обратная матрица  $H_{2^n}^{-1}$  со свойствами:

$$\begin{pmatrix} x_1 \\ \dots \\ x_{2^n} \end{pmatrix} = H_{2^n}^{-1} H_{2^n} \begin{pmatrix} x_1 \\ \dots \\ x_{2^n} \end{pmatrix}$$

Это означает, что спектр Уолша обратим - возможно к получившемуся ряду применить обратное преобразование Уолша и восстановить исходную функцию.

Если убрать (занулить) небольшие коэффициенты в спектре, то получившееся после обратного преобразования Уолша изображение не будет существенно отличаться от исходного. Хотя будет иметь очень немного ненулевых значений, и может быть эффективно сжато в спектральном представлении:

Код 80. Восстановление изображения обратным преобразованием после удаления части значений в спектре Уолша

```
from sympy import ifwht
ff=np.array(spec)
print('removing: ',ff[np.abs(ff)<4].shape[0]/ff.shape[0],' values ')
ff[np.abs(ff)<4]=0
fil=np.array(ifwht(ff)).astype('float')
img=fil.reshape(256,256)
plt.imshow(img,cmap='gray')
```

В данном случае из спектрального представления удалено 31% значений. При удалении 94% значений изображение существенно теряет в качестве, но остается узнаваемым.

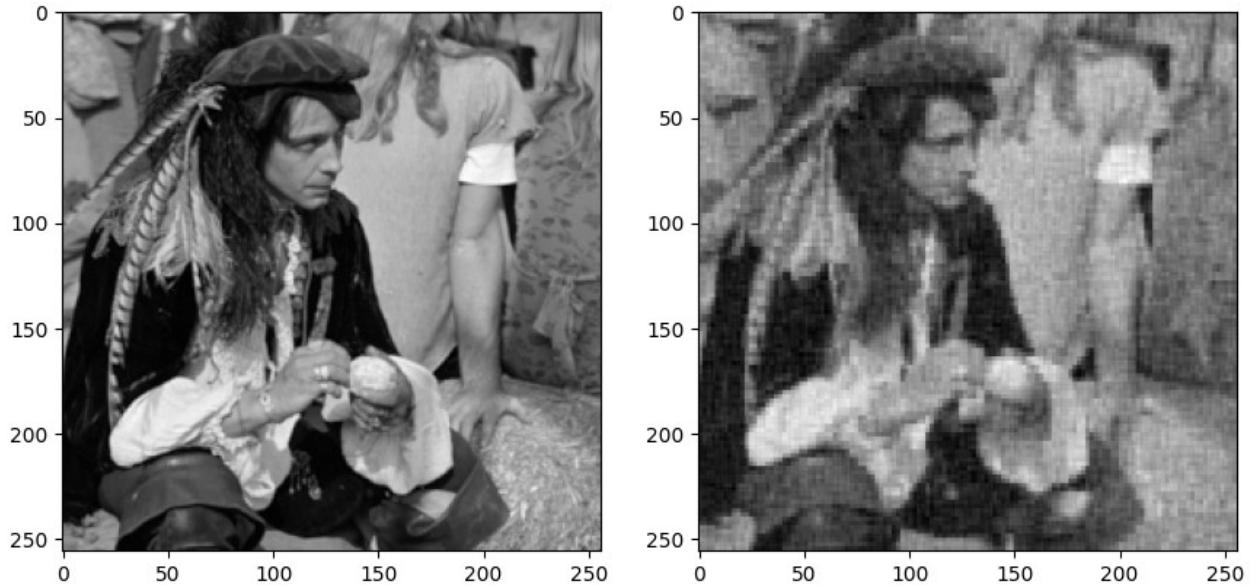


Рис.41. Слева изображение после удаления 31% спектральных составляющих, справа — после удаления 94% составляющих спектра Уолша.

### 8.6.3.Преобразование Фурье

Другим матричным преобразованием является преобразование Фурье (спектр Фурье).

$$A_{r,s} = \sum_{n=1}^N H_{r,s} x_r$$

Базисными функциями преобразования Фурье являются синусы и косинусы, и матрица преобразования составляется из них:

$$H_{r,s} = \frac{1}{\sqrt{N}} e^{-2\pi i(r-1)(s-1)/n}$$

Эта матрица комплекснозначная, поэтому получающиеся составляющие спектра  $A_s$ -комплексные.

Код 81. Преобразование Фурье

```
from scipy.fft import fft
spec=fft(gs_img1D)
plt.plot(spec)
```

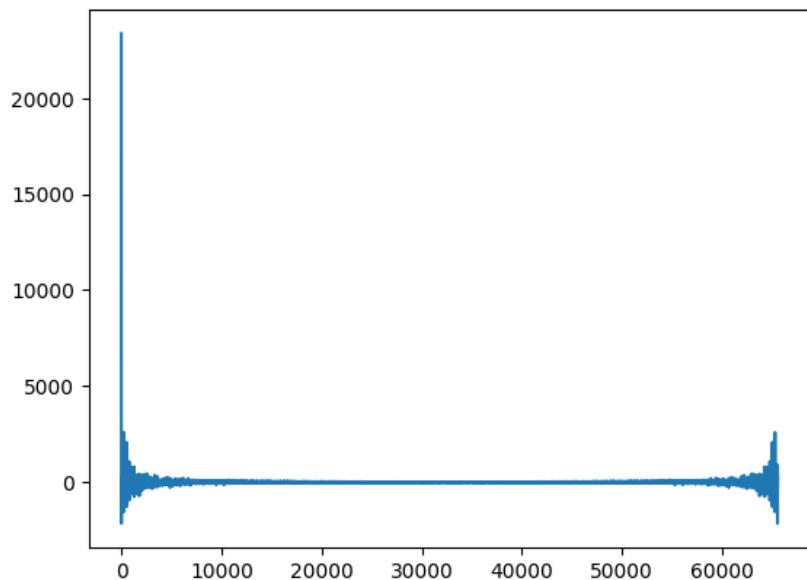


Рис.42. Фурье-спектр тестового изображения

Преобразование также имеет обратное к нему преобразование, поэтому по спектру можно восстановить исходное изображение.

Зануление 32% точек в спектре существенно не изменяет качества изображения. Зануление 94% точек в спектре оставляет изображение узнаваемым.

Код 82. Восстановление изображения обратным преобразованием после удаления части значений в спектре Фурье

```
from scipy.fft import ifft
ff=spec.copy()
print('Remove ',ff[np.abs(ff)<3.5].shape[0]/ff.shape[0],' values')
ff[np.abs(ff)<4]=0
fil=ifft(ff)
img=np.real(fil.reshape(256,256).real)
plt.imshow(img,cmap='gray')
```

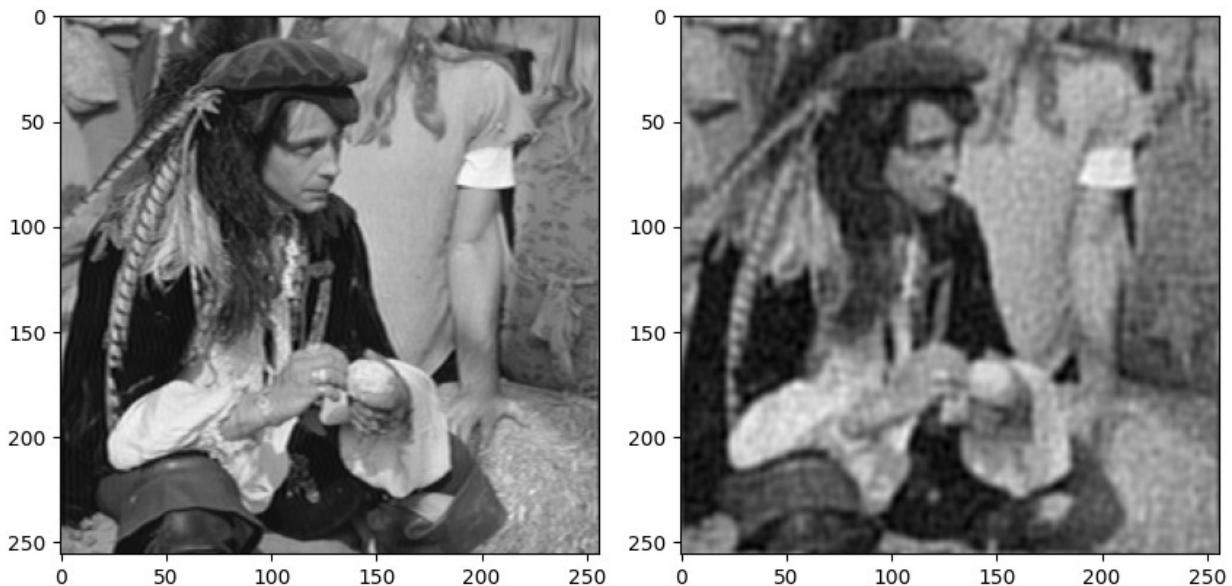


Рис.43. Слева изображение после удаления 32% спектральных составляющих, справа — после удаления 94% составляющих Фурье-спектра

## 8.7.Спектральная оценка периодических функций

Спектральный анализ данных — это мощный инструмент для исследования частотных характеристик сигналов и временных рядов. Основные задачи, которые он решает:

- Обнаружение скрытых периодичностей (например, сезонности);
- Удаление шумов и помех (частотная фильтрация);
- Определение частотной характеристики линейных систем (АЧХ, ФЧХ);
- Использование спектральных методов предсказания (через представление сигналов рядом Фурье);
- Разложение сигнала на гармоники для моделирования и анализа;
- Определение сходства/различия сигналов по их спектрам;
- Эффективное хранение информации в частотной области (например, в аудио и видео кодеках);
- Выявление нехарактерных частотных компонент (например, в мониторинге оборудования или объектов);
- Анализ нестационарных квазипериодических сигналов (например, при обработке речи или музыки).

Одной из основных задач спектрального анализа является поиск числа спектральных составляющих (максимумов в спектре), определение частоты (положения) каждой спектральной составляющей и ее амплитуды, разделение близко расположенных спектральных составляющих.

Код 83. Тестовая функция

```
import matplotlib.pyplot as pyplot
from spectrum import marple_data
data=marple_data
pyplot.plot(marple_data)
```

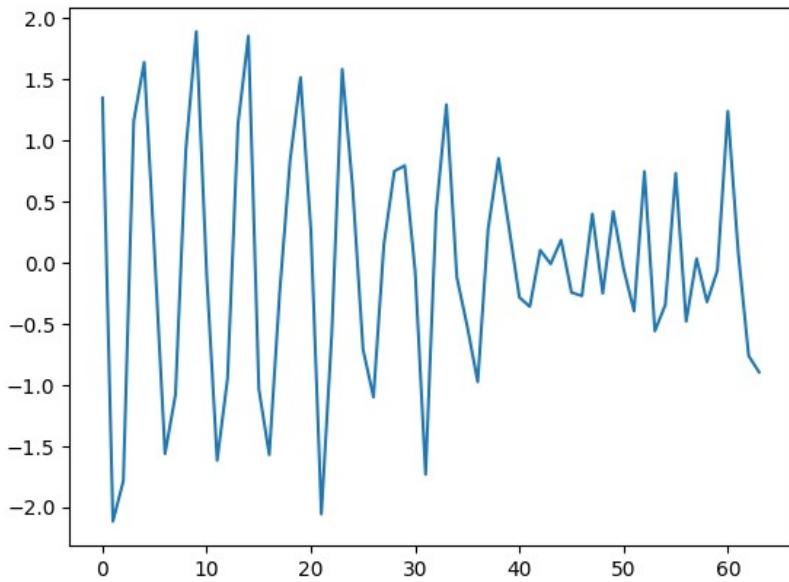


Рис.44. Тестовая функция для анализа

### 8.7.1.Оконное преобразование Фурье

Оконное преобразование Фурье - это метод спектрального анализа, при котором в ряд (интеграл) Фурье разлагается не сама функция  $U(t)$ , а функция, амплитудно-модулированная заданной функцией — окном  $W(t)$ :

$$\tilde{U}(f) = \int e^{-2\pi ift} W(t) U(t) dt$$

Использование оконного преобразования позволяет уменьшить количество фиктивных максимумов в спектре (артефактов, алиасов), связанных с конечностью окна анализа.

Наиболее распространенными типами используемых окон являются — прямоугольное, треугольное (Бартлетта), гауссово, Ханна, Хэмминга, Блэкмена, и Кайзера.

**Код 84. Вычисление оконного преобразования Фурье с окнами Кайзера и прямоугольным**

```
from spectrum import Periodogram
from spectrum.window import Window
pg = Periodogram(marple_data, sampling=1024)
pyplot.yscale('log')
pg.window='kaiser'
pg.run()
pyplot.xlabel('Frequency(norm.)')
pyplot.plot(pg.psd,label='Windowed FFT');
pg.window='rectangle'
pg.run()
pyplot.plot(pg.psd,label='FFT')
pyplot.legend();
```

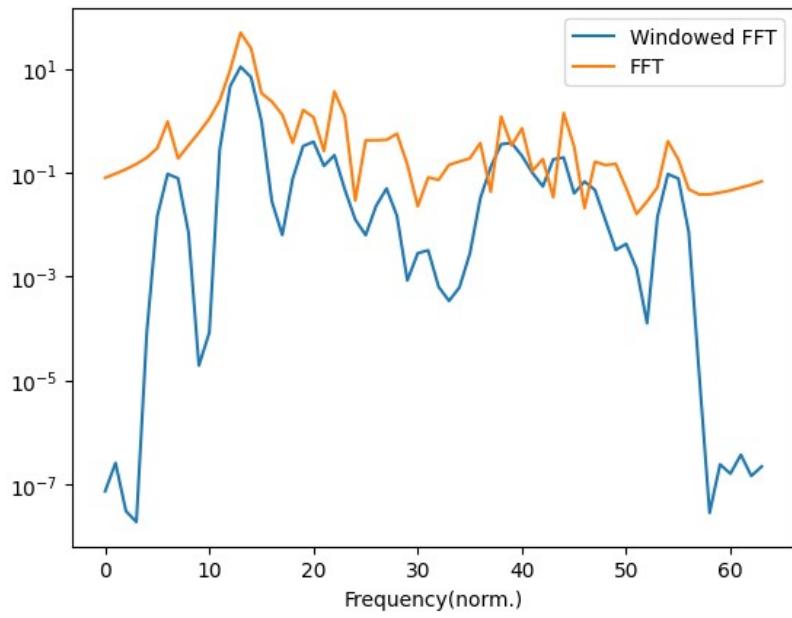


Рис.45. Оконное преобразование Фурье тестового ряда с окном Кайзера

Видно, что оконное преобразование Фурье с окном Кайзера уменьшает число фиктивных пиков, но делает пики шире по сравнению с обычным преобразованием Фурье.

**Код 85. Форма и спектр окна Кайзера**

```
from spectrum.window import Window
pg.window='kaiser'
w = Window(N, pg.window)
w.plot_time_freq()
pg.run()
```

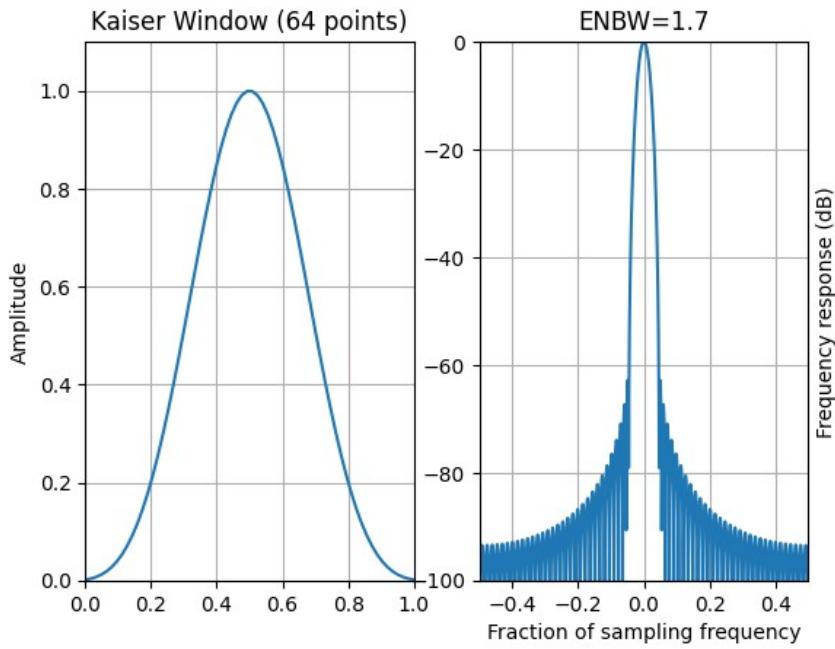


Рис.46. Форма окна Кайзера и боковые лепестки в его спектре - «алиасы»

Важными параметрами окна являются эффективное уширение пика (ENBW) по сравнению с классическим преобразованием Фурье и уровень первого бокового лепестка. Для окна Кайзера уширение пика составляет порядка 1.7, а уровень первого лепестка порядка -60дБ. Это дает в 10000 раз меньшие алиасы по сравнению с алиасами при обычном преобразовании Фурье (т.е. прямоугольном окне) (-15дБ).

### 8.7.2.Параметрический спектр на основе ARMA

Другим способом спектральной оценки является параметрическая спектральная оценка на основе модели ARMA. Поскольку модель ARMA авторегрессионная, то ее спектр представляет собой рациональную функцию, имеющую максимумов не больше чем число коэффициентов регрессии. Такой спектр называется параметрическим.

$$\begin{aligned}
 X_t &= c + \epsilon_t + \sum_{i=1}^p \alpha_i X_{t-i} + \sum_{j=1}^q \beta_j \epsilon_{t-j} \\
 X_t &= \psi(B) W_t = \sum_{i=0}^{\infty} \psi_i W_{t-i} \\
 \psi(B) &= \theta(B)/\phi(B) \\
 f(v) &= \sigma_{\omega}^2 \left| \frac{\theta(e^{-2\pi i v})}{\phi(e^{-2\pi i v})} \right|^2
 \end{aligned}$$

Построим параметрический спектр на основе модели ARMA(15,15). Он будет иметь не более 15 пиков.

#### Код 86. Построение параметрического спектра ARMA(15,15)

```
import numpy as np
from spectrum import arma_estimate, arma2psd
ARcoefs,MAcoefs, resid = arma_estimate(marple_data, 15, 15, 30)
psd = arma2psd(A=ARcoefs, B=MAcoefs, rho=resid, norm=True)
pyplot.yscale('log')
pg.window='kaiser'
pg.run()
pyplot.plot(pg.psd,'--',label='Pg w. Kaiser window')
pyplot.plot(np.linspace(0,N,len(psd)), np.sqrt(psd*N),label='ARMA estimate',c='k')
pyplot.xlabel('Frequency(norm.units)')
pyplot.legend()
```

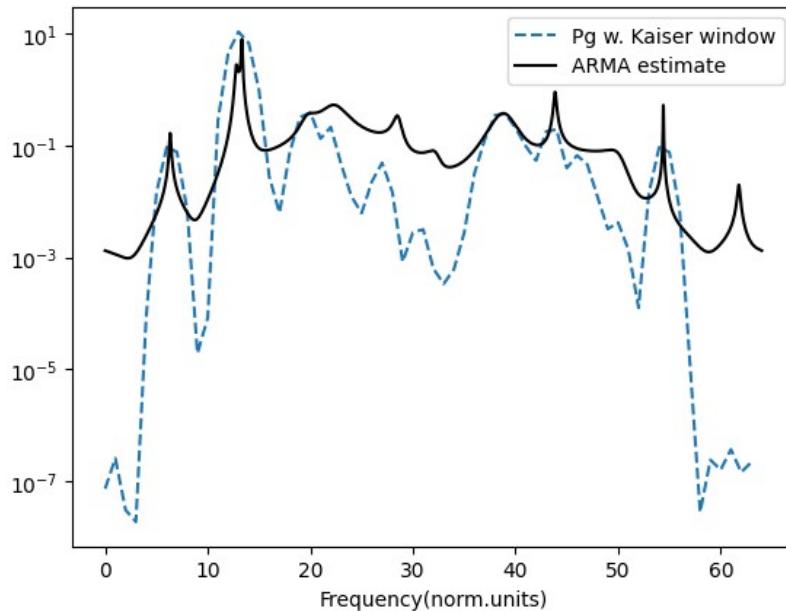


Рис.47. Сравнение спектральной оценки с окном Кайзера и параметрической оценки по модели ARMA(15,15)

Видно что в полученном параметрическом ARMA спектре отсутствуют алиасы, а значит проще детектировать и интерпретировать различные спектральные линии.

#### 8.7.3.Метод MUSIC

Метод MUSIC (также аналогичен методу Писаренко) предполагает, что сигнал состоит из заданного количества гармонических сигналов и аддитивного шума. Частоты этих сигналов определяются через собственные вектора  $\mathbf{v}$  автокорреляционной матрицы  $\mathbf{R}$ .

$$x(n) = \sum_{k=1}^p A_k e^{j(2\pi f_k n + \phi_k)} + w(n)$$

$$R = \frac{1}{N} \sum_{n=0}^{N-1} x(n) x^H(n)$$

$$R = U \Lambda U^H$$

$$P_{MUSIC}(f) = \frac{1}{e^H(f) U_N U_N^H e(f)} = \frac{1}{\sum_{i=p+1}^M [\vec{e}^H \vec{v}_i]^2}$$

$$\vec{e} = [1, e^{i2\pi f}, e^{i2\pi 2f}, \dots, e^{i2\pi Mf}]^T$$

Построим параметрический спектр MUSIC по тестовому датасету:

#### Код 87. Построение параметрического спектра MUSIC

```
from numpy import linalg as LA
import scipy

def MUSIC(data,nfft=4096,p=10):
    m = len(data)
    acf = np.convolve(data,np.conj(data)[::-1])
    center = int(np.ceil(len(acf)/2) - 1)
    Rxx=acf[center:]
    Rx = scipy.linalg.toeplitz(Rxx,np.hstack((Rxx[0], np.conj(Rxx[1:]))))
    D, V = LA.eig(Rx)
    i = np.array(np.argsort(D))
    freq_array = np.arange(-0.5, 0.5, 1/nfft)
    sum_array = np.zeros(nfft, dtype=np.float32)
    for f in np.arange(0, nfft, 1):
        frequencyVector = np.zeros(m).astype(complex)
        for index in np.arange(0, m):
            frequencyVector[index] = np.exp(2 * np.pi * index * freq_array[f] * 1j)
        sum = 0
        for index in np.arange(0, m - p):
            sum = sum + np.abs(
                np.dot(np.transpose(np.conjugate(frequencyVector)), V[:,i[index]]))**2
        sum_array[f] = 1/sum
    return sum_array
music_psd=MUSIC(marple_data,p=15)
pyplot.yscale('log')
pyplot.plot(np.linspace(0,pg.psd.shape[0],music_psd.shape[0]),music_psd,label='MUSIC')
pyplot.plot(pg.psd,'--',label='Pg w. Kaiser window')
pyplot.plot(np.linspace(0,N,len(psd)), np.sqrt(psd*N),label='ARMA estimate',c='k')
pyplot.xlabel('Frequency(norm.units)')
pyplot.legend();
```

Из сравнения видно, что хотя метод MUSIC дает более узкие спектры и часто угадывает, но качество его спектральной оценки по-видимому хуже чем у ARMA. Известно, что метод MUSIC хорошо работает на чисто периодических данных, и намного хуже - на квазипериодических, как наш пример.

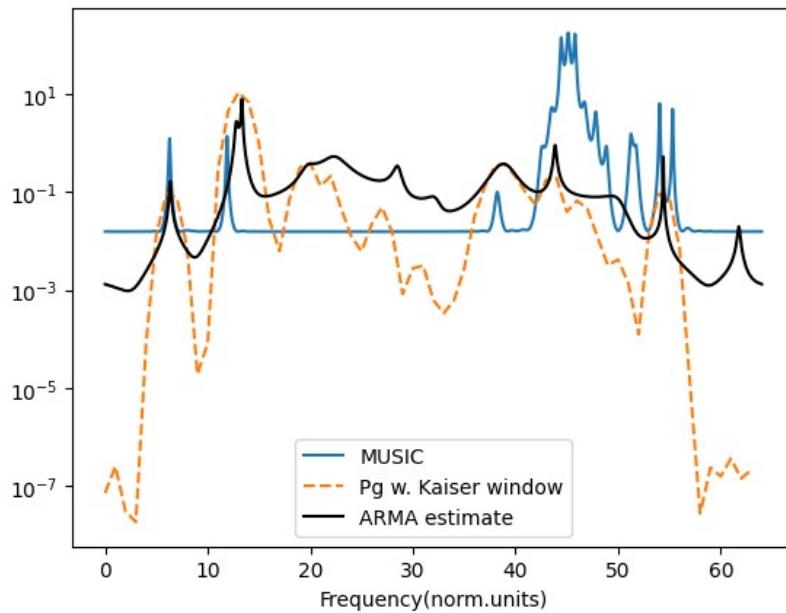


Рис.48. Сравнение методов MUSIC(15), спектра Фурье с окном Кайзера и параметрического спектра ARMA(15,15)

## 8.8. Анализ нестационарных сигналов

### 8.8.1. Спектрограмма

Спектрограмма представляет собой квадрат модуля скользящего оконного преобразования Фурье (STFT):

$$\tilde{U}(\tau, f) = \left| \int \exp(-2\pi i f(t-\tau)) W(t-\tau) U(t) dt \right|^2$$

При этом параметр  $\tau$  часто называют медленным временем.

Спектрограммы используют для анализа нестационарных сигналов, имеющих несколько масштабов изменений — медленный (анализируемый по медленному времени) и быстрый (анализируемый по частотной переменной). Позволяют исследовать изменения быстроменяющихся (высокочастотных) составляющих сигналов во времени.

**Код 88. Построение спектрограммы звукозаписи голосов дельфинов**

```
from spectrum import Spectrogram, dolphin_filename, readwav
print(dolphin_filename)
data, samplerate = readwav(dolphin_filename)
p = Spectrogram(data, ws=128, W=4096, sampling=samplerate)
p.periodogram()
p.plot()
```

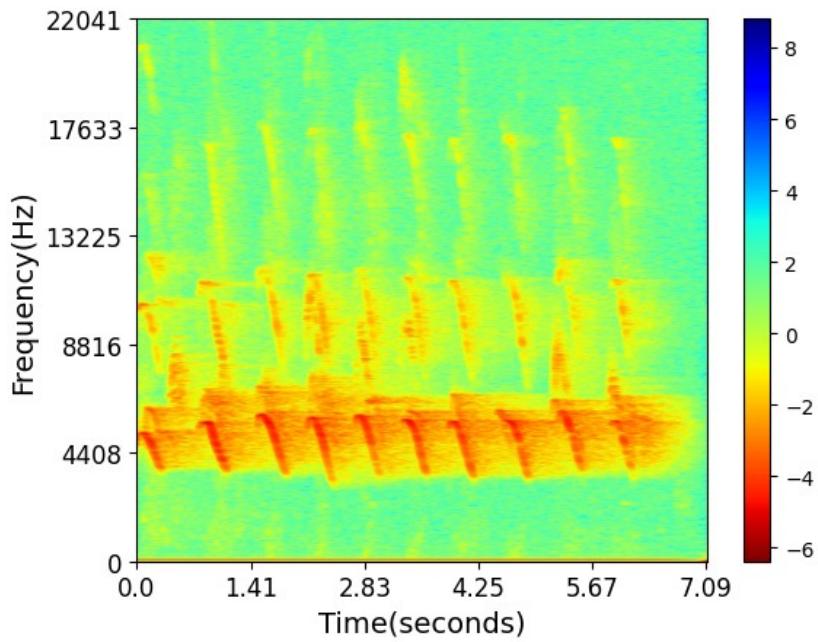


Рис.49. Пример спектрограммы звукозаписи голосов дельфинов

### 8.8.2. Вейвлет-преобразование

Вейвлет-преобразование  $T(a,b)$  последовательности  $x(t)$  это преобразование вида:

$$T(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} x(t) \psi\left(\frac{t-b}{a}\right) dt$$

где  $\psi$  - некая функция с конечным носителем и нулевым средним - вейвлет. Параметр  $b$  называется времененным сдвигом, а параметр  $a$  - параметром растяжения (масштабом).

Существует большое количество различных вейвлетов, от выбора которых существенно зависит результат преобразования.

Приведем пример вейвлета.

Код 89. Пример вейвлета Морлета (Morlett)

```
import pywt
wavelet, time = pywt.ContinuousWavelet('morl').wavefun(length=300)
plt.plot(wavelet)
```

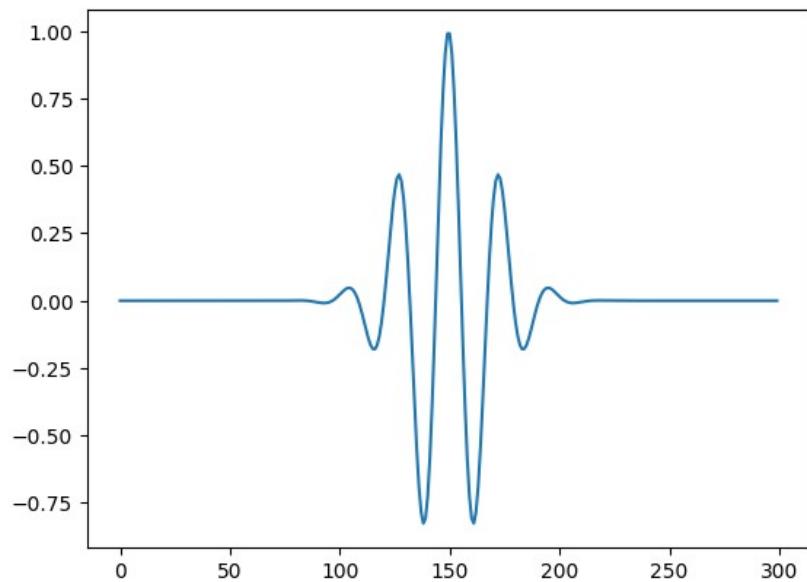


Рис.50. Пример вейвлета Морлета (Morlett)

Приведем пример вейвлет-преобразования звукозаписи голосов дельфинов

Код.90. Пример вейвлет-преобразования звукозаписи голосов дельфинов (вейвлет Morlett)

```
import pywt
scales = np.arange(1, 512, 4)
coefficients, freqs = pywt.cwt(data[0:60000:3], scales, 'morl')
coefficients=np.log(np.abs(coefficients)+1e-10)
plt.imshow(coefficients, aspect=60)
plt.ylabel('Period(norm.units)')
plt.xlabel('Время(norm.units)')
```

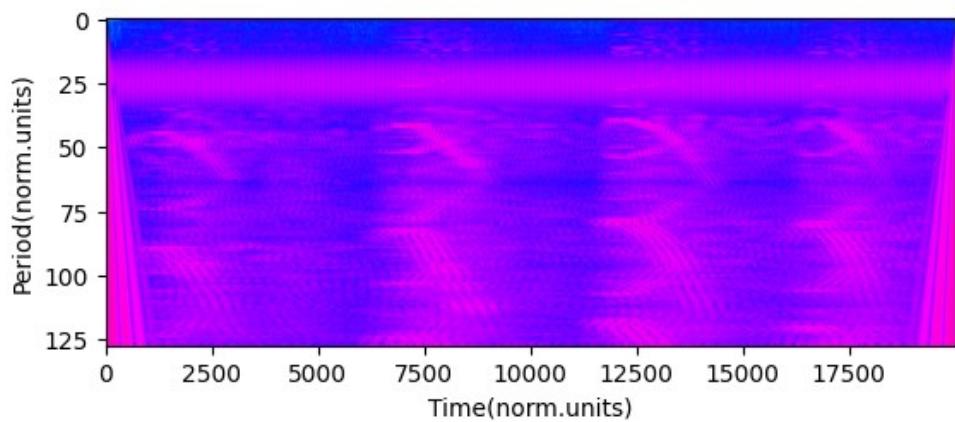


Рис.51. Пример вейвлет-преобразования звукозаписи голосов дельфинов (вейвлет Morlett)

Плюсы и минусы использования преобразований приведены в таблице.

	<b>Вейвлет-преобразование</b>	<b>Спектrogramма Фурье</b>
Плюсы	<ul style="list-style-type: none"> <li>-Фиксированное соотношение между спектральным и временным разрешением на всех масштабах</li> <li>-Минимум алиасов</li> </ul>	<ul style="list-style-type: none"> <li>-Понятность</li> <li>-Быстрые алгоритмы</li> <li>-Стандартизованность</li> <li>-Связь с фундаментальными решениями</li> </ul>
Минусы	<ul style="list-style-type: none"> <li>-Сложнее интерпретировать масштаб.</li> <li>- Вычислительно затратное</li> <li>- Произвольность вейвлетов</li> </ul>	<ul style="list-style-type: none"> <li>-Утечки спектра (алиасы).</li> <li>-Произвольность окна для борьбы с алиасами</li> <li>-Нет фиксированного соотношения между спектральным и временным разрешением на всех частотах</li> </ul>

## 9. Извлечение признаков.

Иногда встречаются задачи с тысячами или даже миллионами параметров для каждого обрабатываемого объекта. Допустим, вы анализируете поток видео, стандартное разрешение кадра —  $1900 \times 1000$  пикселей, что дает около 2 миллионов параметров для каждого кадра. Данных очень много, и важно определить, какие признаки существенны, а какие нет. Это помогает ускорить работу с данными, сделать алгоритмы быстрее и уменьшить объем хранимой информации. Эта задача называется “извлечением признаков” (Feature Extraction).

Методы можно разделить на методы без генерации новых признаков и методы с генерацией новых признаков.

Детально можно посмотреть работу [8].

### 9.1. Методы без генерации новых признаков

Как определить, какие признаки важны, а какие нет, без генерации новых признаков? Очевидно, что в таком случае задача сводится к понижению размерности входных данных – отбору существенных для решения задачи признаков и отсеву несущественных.

Задачи понижения размерности можно решать тремя основными методами:

1. Одномерные методы предполагают, что целевая переменная зависит только от одного признака;
2. Многомерные методы учитывают влияние нескольких признаков на результат;
3. Методы на основе модели используют такую обучаемую на данных модель для отбора оптимальных признаков.

#### 9.1.1. Одномерные методы

Начнем с одномерных методов. Здесь мы оцениваем зависимость целевой переменной  $Y$  от каждого признака по отдельности. Поведение каждого признака сравнивается с поведением целевой функции, и таким образом выделяются самые важные признаки.

Для поиска можно использовать:

- Максимум коэффициента корреляции Пирсона, который измеряет силу линейной зависимости;
- Максимум ранговой корреляции Спирмена, который оценивает силу монотонной зависимости;
- Максимум информационных мер, которые анализируют связь между величинами часто в виде условных вероятностей, и особенно полезны для дискретных данных.

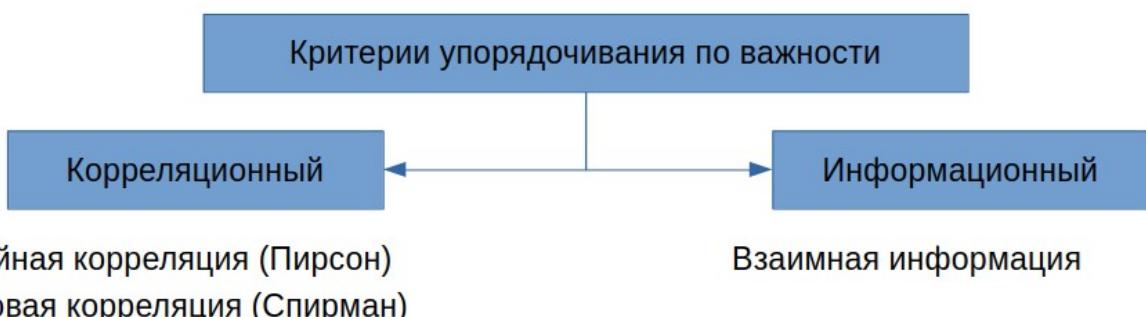


Рис.52. Одномерные методы

Корреляцию Пирсона и Спирмена мы обсуждали ранее и помним, что она применима для оценки связи между числовыми или упорядоченными категориальными величинами.

Метод взаимной информации (Mutual Information) оценивает, насколько знание одной переменной уменьшает неопределенность другой, и при поиске признака старается максимизировать функционал:

$$MI(X;Y)=\sum P(x,y)\cdot \log \frac{P(x,y)}{P(x)\cdot P(y)}$$

Для непрерывных переменных вместо суммы используется интеграл.

#### Код 91. Одномерное извлечение признаков с помощью коэффициента Спирмена

```
import numpy as np
from scipy.stats import spearmanr
x=np.linspace(-10,10,1000)
X=np.stack((x,x**2,np.sin(x),np.cos(x)),axis=-1)
Y=5*X[:,0]+3*X[:,2]+5*X[:,3]**2+np.random.randn(X.shape[0])*3
i_best,ssc_best=0,0
for i in range(X.shape[1]):
    ssc,pv=spearmanr(X[:,i],Y)
    if(abs(ssc)>abs(ssc_best)):
        ssc_best=ssc
        i_best=i
print(i_best,ssc_best)
#
0 0.9902996102996102
```

Видно, что наиболее важным с точки зрения коэффициента Спирмена можно считать нулевой признак, который на 99% коррелирует с целевой функцией.

### 9.1.2. Многомерные (жадные) методы

Недостатком одномерных методов является то что они не учитывают взаимодействие признаков: например зависимость от суммы или произведения признаков.

Многомерные методы позволяют выявлять сложные зависимости между целевой функцией и входными признаками. Однако для этого необходимо найти способ выявлять зависимости именно от комбинаций признаков. Существует несколько основных подходов к поиску комбинаций: полный перебор, жадное добавление, жадное удаление и комбинированные методы.

Методы, основанные на полном переборе, предлагают следующий подход: перебрать все возможные комбинации признаков и найти минимальную комбинацию, при которой результат прогноза целевой функции наиболее качественный. Для каждого набора признаков мы обучаем модель (например, решающее дерево или нейронную сеть) и оцениваем точность предсказания. Комбинация, дающая наилучший результат, считается оптимальной, а остальные признаки могут быть отброшены. Это самый надежный, но самый медленный способ. Если у нас имеется  $N$  признаков, то количество возможных комбинаций составляет  $2^N - 1$ . Уже при 20 признаках количество комбинаций становится неприлично большим (более миллиона).

Более быстрым способом поиска являются жадные алгоритмы. Самый простой из них - жадное добавление признаков. Сначала мы находим один наилучший признак, затем пытаемся добавить к нему следующий, выбирая тот, который максимально улучшает

качество модели в комбинации этих двух признаков. После этого ищется третий признак и так далее. Этот процесс продолжается до тех пор, пока добавление новых признаков не перестает улучшать результат.

#### Код 92. Многомерное извлечение признаков с помощью жадного добавления

```
from sklearn.feature_selection import SequentialFeatureSelector  
from sklearn.linear_model import LinearRegression  
lr=LinearRegression()  
sfs = SequentialFeatureSelector(lr, n_features_to_select=2, direction='forward')  
sfs.fit(X, Y)  
sfs.get_support()  
#  
array([ True, False,  True, False])
```

Видно, что при жадном добавлении наиболее важными признаками были найдены 0й и 2й признак.

Другой способ — это жадное удаление признаков. Мы начинаем с полного набора признаков и последовательно удаляем каждый признак и проверяем, не улучшится ли качество модели по остальным признакам. Если после удаления какого-то признака качество возрастает (не убывает), значит, этот признак был избыточным.

#### Код 93. Многомерное извлечение признаков с помощью жадного удаления

```
sfs = SequentialFeatureSelector(lr, n_features_to_select=2, direction='backward')  
sfs.fit(X, Y)  
sfs.get_support()  
#  
array([ True, False,  True, False])
```

Видно, что при жадном добавлении наиболее важными признаками были также найдены 0й и 2й признак.

### 9.1.3.Методы на основе моделей

Еще один подход - методы на основе моделей. Мы предполагаем что данные у нас описываются определенной моделью (линейная регрессия, случайный лес, нейронная сеть) и анализируем ее коэффициенты, архитектуру, и/или качество работы модели. Для поиска признаков можно использовать как одномерные, так и многомерные методы.

#### 9.1.3.1.Комбинированное жадное добавление/удаление

Комбинированный метод включает в себя два этапа, чередуемых последовательно — сначала жадно добавить признак, после чего жадно удалить признаки. После чего эти действия повторить. Метод проиллюстрирован ниже.

#### Код 94. Многомерное извлечение признаков с помощью комбинированного жадного добавления/удаления

```
from sklearn.linear_model import LinearRegression  
from sklearn.metrics import r2_score
```

```

def select_important_features(X, y, target_features=3):
    lr = LinearRegression()
    selected_features = []
    for _ in range(target_features):
        best_feature = None
        best_score = -np.inf
        for feature in range(X.shape[1]):
            #жадное добавление
            if feature in selected_features: continue
            temp_features = selected_features[:] + [feature]
            lr.fit(X[:, temp_features], y)
            score = r2_score(y, lr.predict(X[:, temp_features]))
            if score > best_score:
                best_score = score
                best_feature = feature
        selected_features.append(best_feature)
    #жадное удаление
    if len(selected_features) > 1:
        worst_feature = None
        best_removal_score = -np.inf
        for feature in selected_features:
            reduced_features = [f for f in selected_features if f != feature]
            lr.fit(X[:, reduced_features], y)
            score = r2_score(y, lr.predict(X[:, reduced_features]))
            if score > best_removal_score:
                best_removal_score = score
                worst_feature = feature
        if best_removal_score >= best_score:
            selected_features.remove(worst_feature)
    return selected_features[:target_features]

print("Лучшие признаки:", select_important_features(X, Y, 3))
#
Лучшие признаки: [0, 2, 1]

```

### 9.1.3.2.Модель линейной регрессии

В случае линейной регрессии незначимые признаки будут иметь коэффициенты, близкие к нулю. Таким образом, по величине коэффициентов можно оценить, насколько важен признак.

Необходимо заметить, что если каждый признак обладает своим среднеквадратичным отклонением, важность признака будет определяться произведением коэффициента регрессии и среднеквадратичного отклонения этого признака.

#### Код 95. Многомерное извлечение признаков с помощью линейной регрессии

```

lr=LinearRegression().fit(X,Y)
Yp=lr.predict(X)
print(lr.coef_*X.std(axis=0))
#
[28.89348829 0.45073578 2.09096558 0.10380632]

```

Видно, что наиболее важным с точки зрения линейной регрессии является нулевой

признак, а за ним по важности следует 2й.

Преимущество таких моделей в том, что они дают быстрый и понятный результат - коэффициенты прямо показывают влияние каждой переменной. Однако есть и недостаток: если тип модели не соответствует реальной зависимости в данных (например если зависимость нелинейна), то важные признаки могут быть пропущены.

### 9.1.3.3.Модель решающих деревьев

Одним из более универсальных модельных методов является метод на основе решающих деревьев.

Критерий информативности в деревьях показывает, насколько хорошо разделяются классы при разбиении по конкретному признаку. Мы анализируем:

1. Уменьшение энтропии или индекса Джини после каждого разбиения
2. Количество точек, попавших в каждую ветвь
3. Суммарное улучшение критерия по всем разбиениям с данным признаком

Чем больше это улучшение, тем важнее признак.

#### Код 96. Многомерное извлечение признаков с помощью дерева

```
from sklearn.tree import DecisionTreeRegressor  
rg=DecisionTreeRegressor().fit(X,Y)  
print(rg.feature_importances_)  
#[0.8797 0.1003 0.0027 0.0171]
```

Видно, что с точки зрения более сложной модели наиболее важными являются 0й и 1й признак.

Однако одно дерево может давать неточные оценки из-за переобучения. Поэтому лучше использовать ансамбли моделей (например, случайный лес), где важность признаков усредняется по множеству деревьев.

#### Код 97. Многомерное извлечение признаков с помощью случайного леса

```
from sklearn.ensemble import RandomForestRegressor  
rg=RandomForestRegressor().fit(X,Y)  
print(rg.feature_importances_)  
#[0.8553 0.1312 0.0039 0.0093]
```

Видно, что с точки зрения более сложной модели случайного леса наиболее важными являются 0й и 1й признак.

### 9.1.3.4.Перестановочный метод

Предыдущие методы были основаны на фиксированной модели — линейной регрессии или дерева решений, и они не могут применяться для других моделей. Что делать, если модель сложная, например нейронная сеть? Есть более простой и общий метод оценки важности — перестановочный метод:

1. Обучаем дерево, или случайный лес или любую другую модель, дающую хороший прогноз;
2. На тестовой выборке перемешиваем значения одного столбца;

3. Смотрим, насколько ухудшилась метрика качества прогноза.

Если качество сильно упало - признак важен, если нет - его влияние незначительно. В многомерном случае используем жадные варианты метода.

Код 97. Многомерное извлечение признаков с помощью перестановочного метода

```
from sklearn.inspection import permutation_importance
fi = permutation_importance(rg, X, Y, n_repeats=30)
print(fi['importances_mean'])
#[1.8739 0.1543 0.0042 0.0085]
```

Очевидно, что результат работы перестановочным методом на случайному лесе совпал с определением важности по коэффициенту Джини: наиболее важными остаются 0й и 1й признак.

## 9.2.Методы с генерацией новых признаков

Следующий шаг — выявление значимых переменных для снижения размерности. Мы уже говорили, что извлекаем зависимости между выходными значениями и компонентами входного вектора, но что делать, если зависимость сложная? Теорема Джонсона-Линденштрауса утверждает, что для сильно многомерных входных векторов можно построить пространство меньшей размерности, где расстояния между точками почти сохраняются. Например, трёхмерные точки можно отобразить в двумерное пространство с минимальными искажениями расстояний между соответствующими точками. Эта теорема позволяет эффективно снижать размерность, сохраняя структуру данных.

Нам только нужно построить какую-то модель, которая преобразует входные вектора в новые компоненты оптимальным образом так, чтобы новых компонент было небольшое число. Если модель выбрана неправильно - зависимость не извлечется. Существуют статистические методы, которые позволяют автоматически строить удачные модели. Они делятся на линейные и нелинейные.

Рассмотрим задачу кластеризации ирисов (датасет Фишера). Признаки в датасете трехмерны, поэтому стоит проблема отображения их наилучшим образом на двумерной плоскости

Код 98. Тестовый датасет ирисов Фишера

```
from sklearn import datasets
import matplotlib.pyplot as plt
iris = datasets.load_iris()
fig,axs=plt.subplots(1,3,figsize=(10,3))
k=0
for i in range(3):
    for j in range(i+1,3):
        axs[k].scatter(iris.data[:,i],iris.data[:,j],c=iris.target)
        k+=1
```

Выбирая разные признаки мы видим, что разные комбинации признаков по-своему отображают проблему разделимости. Хуже всего иллюстрирует различия в типах ирисов крайне левая пара.

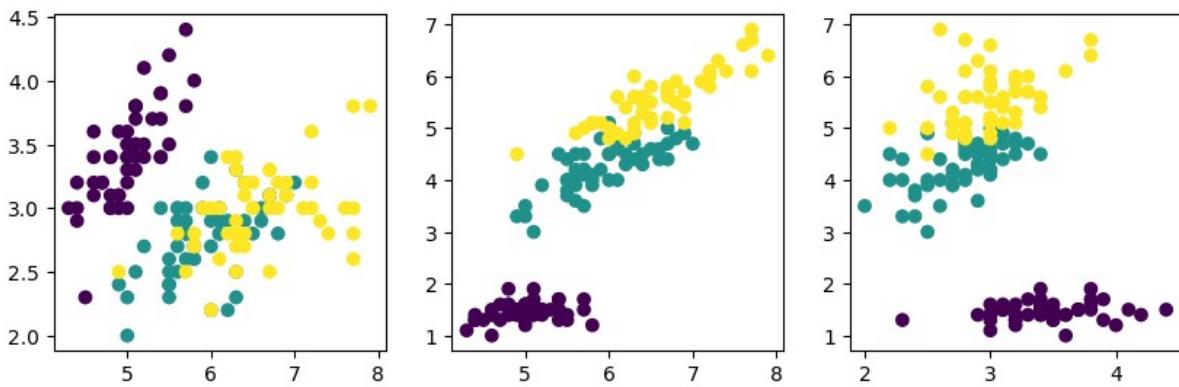


Рис.53. Датасет ирисов в различных проекциях. Цвет означает тип ириса.

Найдем оптимальное представление для этих данных, чтобы различия были наиболее сильными.

### 9.2.1.Линейные методы

Линейные методы предлагают преобразовать входные признаки так, чтобы новые признаки были линейными комбинациями старых. Эти методы применимы, когда зависимость целевой функции от переменных линейная.

Фактически задача сводится к матричному разложению. Матричное разложение это представление набора входных векторов в виде произведения матриц. Чаще всего используются произведения двух или трех матриц, хотя в принципе матриц может быть больше - такой процесс называется факторизацией.

#### 9.2.1.1.Метод случайных проекций

Первый метод для решения этой задачи — метод случайных проекций. Он эквивалентен умножению матрицы входных векторов  $X$  на некую случайную матрицу  $W$ , то есть факторизации вида:

$$X = W^{-1} Y$$

здесь  $W$  — случайная матрица, а  $Y$  — новые проекции.

Фактически метод предлагает случайно выбрать коэффициенты преобразования  $W$  с помощью генератора случайных чисел (например, из нормального распределения с нулевым средним):

$$y_{i,j} = \sum_{k=1}^D W_{j,k} x_{i,k}$$

Или в матричном виде:

$$Y = W X$$

Здесь  $y$  — новые переменные,  $x$  — старые переменные, веса  $W$  генерируются однажды и случайно.

Если коэффициенты  $W$  выбраны удачно, то разделение точек в новом пространстве получится лучше, чем в исходном пространстве. В библиотеке Scikit-learn есть готовая реализация этого метода.

Код 99. Преобразование данных методом случайных проекций

```

from sklearn import random_projection
transformer = random_projection.GaussianRandomProjection(n_components=2,random_state=42)
X_new = transformer.fit_transform(iris.data)
plt.scatter(X_new[:,0],X_new[:,1],c=iris.target);

```

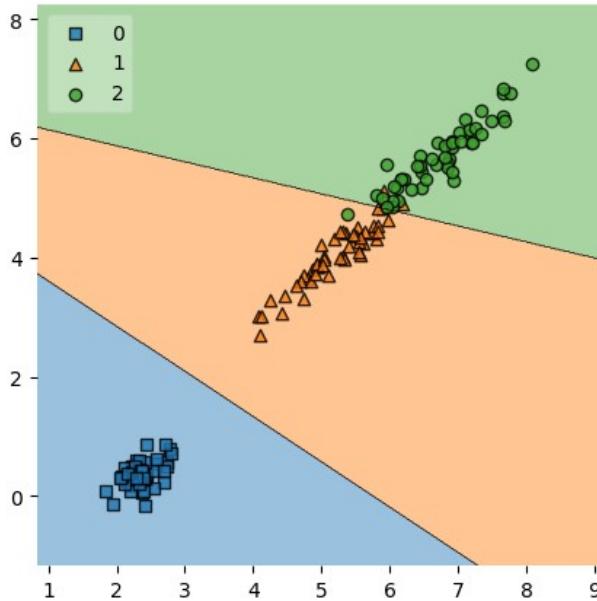


Рис.54. Применение метода случайных проекций к ирисам. Цвет означает тип ириса. Линиями обозначены возможные границы разделения ирисов на классы.

Метод пытается спроецировать данные в новое пространство с заданной размерностью. На входе у нас 3 измерения, а на выходе получается размерность 2. Оказалось, что метод работает неплохо - мы получаем точки, которые достаточно легко отличить друг от друга.

### 9.2.1.2.Метод главных компонент

Теперь рассмотрим более сложный метод - метод главных компонент (PCA). Его суть в том, что веса  $W$  выбираются не случайно, а через решение задачи оптимизации. Самая простая постановка - найти такие оси, чтобы данные были максимально "вытянуты" вдоль них.

Представьте облако точек данных: мы проводим первую ось в направлении наибольшего разброса точек (где дисперсия положений максимальна). Это удобно, потому что вдоль такой оси точки будут максимально разнесены.

После выделения первой главной компоненты мы переходим к оставшимся измерениям. В этом подпространстве мы повторяем процедуру, снова находя направление с максимальной дисперсией. Это позволяет последовательно выделять наиболее значимые направления в данных. Решение сводится к нахождению сингулярного разложения матрицы данных  $X$ :

$$X = U \Sigma Y$$

Получается представление, где матрицы  $U, Y$  являются матрицами поворота (их определители равны 1), а диагональная матрица  $\Sigma$  содержит сингулярные числа, отражающие важность каждой компоненты. Эти числа упорядочены по убыванию: первое соответствует направлению с максимальной дисперсией, второе - следующему по

значимости, и так далее. Это позволяет ранжировать компоненты по важности и при необходимости отбрасывать менее значимые.

Смысл метода в том, что любое линейное преобразование можно представить как два поворота и одно растяжение: сначала поворачиваем систему координат, затем растягиваем вдоль каждой новой оси по-своему, потом снова поворачиваем.

Тогда новые координаты выражаются линейно через старые:

$$Y = (U \Sigma)^{-1} X$$

Посмотрим реализацию этого метода:

#### Код 99. Преобразование данных методом главных компонент

```
from sklearn.decomposition import PCA  
pca = PCA(n_components=2)  
new_data=pca.fit_transform(iris.data)  
plt.scatter(new_data[:,0],new_data[:,1],c=iris.target)
```

Мы хотим выделить только две главные компоненты, чтобы визуализировать данные на плоскости. Когда мы это делаем и рисуем график, видим, что точки начинают компактно группироваться в отдельные облака и допускают последующую классификацию или кластеризацию.

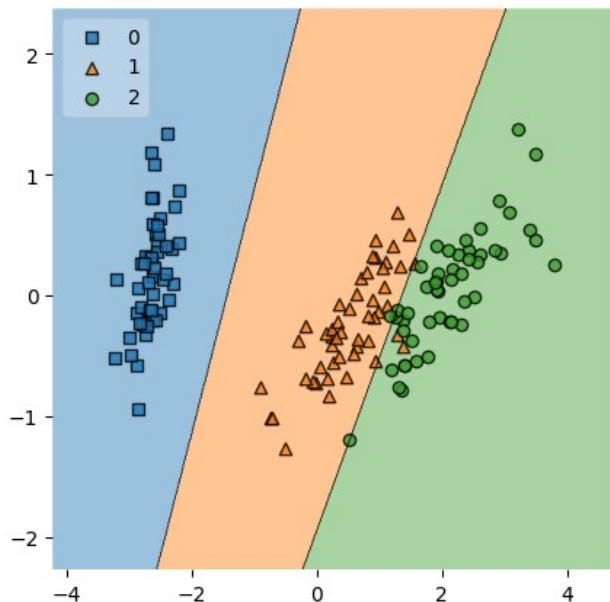


Рис.55. Применение метода главных компонент к ирисам. Цвет означает тип ириса. Линиями обозначены возможные границы разделения.

#### 9.2.1.3. Неотрицательное матричное разложение

Самый простой метод факторизации - разложение на две матрицы, использованный в методе случайных проекций. Однако, иногда встречаются случаи, когда все признаки неотрицательны (или имеют одинаковый знак), и требуется сохранить это свойство в новом пространстве. Например, когда нужно разложить неотрицательные данные - например доходы населения по регионам или количество откликов посетителей на разные товары. Эти

данные по своей природе не могут быть отрицательными, поэтому для интерпретируемости результатов важно, чтобы все элементы матриц разложения (и соответственно данные в новой системе координат) были неотрицательными. Тогда удобно применять метод неотрицательной матричной факторизации - оптимальную реализацию матричной факторизации на две матрицы с неотрицательными коэффициентами.

В этом случае мы имеем матричное разложение:

$$X \approx UY$$

где нужно минимизировать норму Фробениуса - сумму квадратов отклонений между моделью и экспериментальными значениями X:

$$\|X - UY\|^2 = \min$$

при ограничениях:

$$U_{i,j} \geq 0, Y_{i,j} \geq 0$$

Чтобы исключить очевидное решение  $Y = X$  необходимо, чтобы размерность новых векторов Y была меньше, чем размерность исходных признаков X.

Решение этой задачи оптимизации обычно сводится к итеративному процессу. Фактически мы применяем метод наименьших квадратов: вычисляем градиент целевой функции и приравниваем его к нулю по всем переменным. Это приводит к системе уравнений, которая может быть не совсем линейной. Альтернативно можно использовать градиентный спуск, где на каждом шаге вычисляется ошибка (отклонение модели от эксперимента) и умножается на вектор градиента.

Тогда новые координаты выражаются линейно через старые:

$$Y = U^{-1}X$$

Основной параметр при обучении - learning rate ( $\lambda$ ), который является гиперпараметром. Главный недостаток градиентного спуска - медленная скорость работы, так как на каждом шаге нужно суммировать по всем точкам датасета, который может быть очень большим. Поэтому часто используют стохастический градиентный спуск, где берётся только один случайный элемент вместо полной суммы. Более эффективным является метод чередующихся наименьших квадратов (ALS) - попеременно фиксируем одну матрицу и оптимизируем другую (U и Y).

Рассмотрим задачу нахождения оптимальных признаков для датасета MNIST — рукописные цифры.

#### Код 100. Тестовый датасет MNIST

```
digits = datasets.load_digits()
fig, axs=plt.subplots(3,7, figsize=(10,5), constrained_layout=True)
for i in range(21):
    axs[i//7,i%7].imshow(digits.images[i], cmap='gray_r')
    axs[i//7,i%7].set_title(digits.target[i])
    axs[i//7,i%7].set_xticks([])
    axs[i//7,i%7].set_yticks([])
```

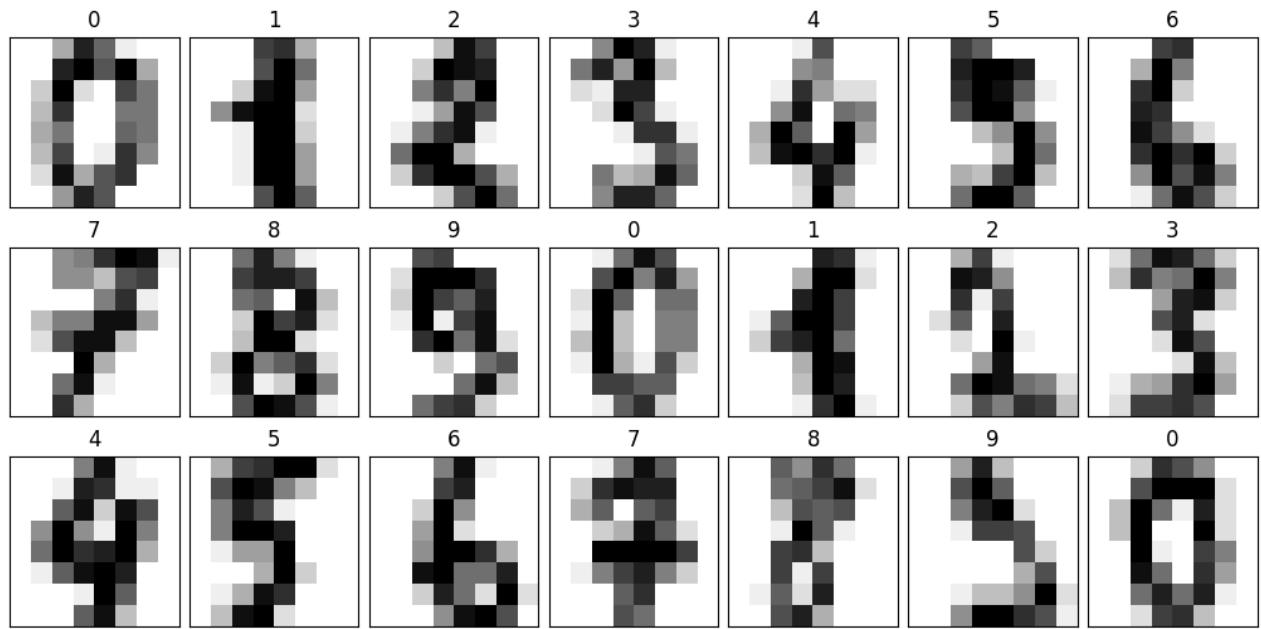


Рис.56. Пример элементов датасета MNIST

Размерность каждого вектора в датасете равна 64, значения каждой компоненты от 0 до 1 (или от 0 до 255 в зависимости от реализации). Найдем оптимальное двумерное представление этих векторов методом NMF.

Код 100. Преобразование данных методом неотрицательного матричного разложения

```
from sklearn.decomposition import NMF
plt.figure(figsize=(5,5))
nmf = NMF(n_components=2, random_state=42).fit(digits.data)
yp=nmf.transform(digits.data)
plt.scatter(yp[:,0],yp[:,1],c=digits.target,s=3);
```

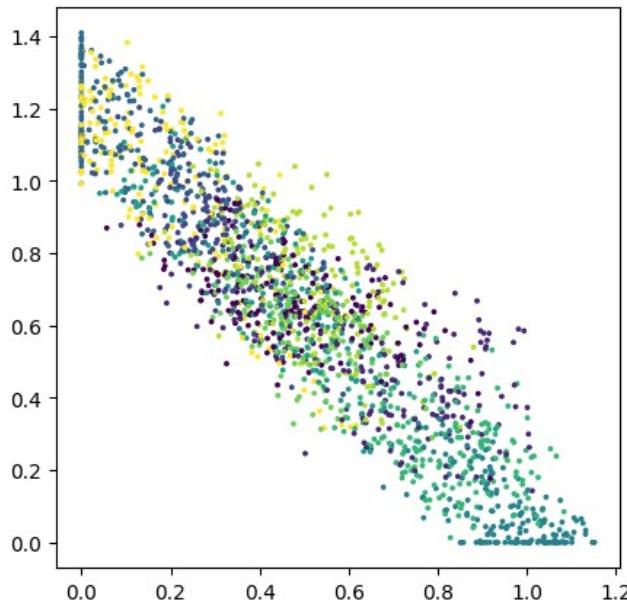


Рис.57. Применение метода NMF к рукописным цифрам. Цвет означает цифру.

Важно понимать, что в отличие от метода главных компонент, NMF:

- Всегда дает неотрицательные результаты;
- Является итеративным решением;
- Не обеспечивает точного решения, только аппроксимацию

### 9.2.2.Нелинейные методы

Рассмотрим нелинейные методы преобразования. В этом случае зависимость между координатами в старой и новой системах координат будет нелинейной, и наши данные (изображения) окажутся "разбросаны" по разным участкам плоскости. Существуют такие преобразования координат, которые позволяют существенно разнести различные изображения по разным углам графического поля.

Нелинейные методы начинают эффективно работать, когда зависимость целевой функции от входных переменных очень сложная и её нельзя аппроксимировать линейной связью, как в случае линейных методов.

Нелинейные методы основаны на том, чтобы подобрать новое пространство, которое зависит от исходного нелинейно. То есть новые координаты уже не являются линейной комбинацией исходных, а вводятся в соответствии с некоторыми условиями.

#### 9.2.2.1.Многомерное шкалирование(MDS)

Первый нелинейный метод, который стоит рассмотреть — многомерное шкалирование (MDS). Идея метода в следующем: давайте введём новые точки в пространстве меньшей размерности (например, на плоскости) и расставим их так, чтобы расстояния между ними примерно соответствовали расстояниям в исходном 64-мерном пространстве. Человеческое восприятие ограничено 2D–3D (иногда 4D, если учитывать время), поэтому обычно данные проецируют на плоскость или трёхмерное пространство.

В MDS мы берём точки в исходном пространстве X и пытаемся расположить их в новом пространстве Y так, чтобы расстояния между ними  $\rho_y(Y_i, Y_j)$  были как можно ближе к

исходным  $\rho_x(X_i, X_j)$ . Точного совпадения не будет, но мы сделаем так, чтобы ошибка была минимальной.

$$\sum_{i < j} (\rho_y(Y_i, Y_j) - \rho_x(X_i, X_j))^2 \rightarrow \min$$

здесь  $\rho_x(X_i, X_j), \rho_y(Y_i, Y_j)$  - расстояния между точками в старом и новом пространстве, чаще всего расстояния евклидовые. Обычно задача решается подбором положения точек Y до достижения минимального значения.

Давайте применим его к нашему датасету (возьмем 1000 точек для примера и ускорения работы — метод очень медленный).

**Код 101. Преобразование данных методом многомерного шкалирования**

```
from sklearn.manifold import MDS
mds = MDS(n_components=2)
yp=mds.fit_transform(digits.data[:1000])
plt.scatter(yp[:,0],yp[:,1],c=digits.target[:1000],s=20);
```

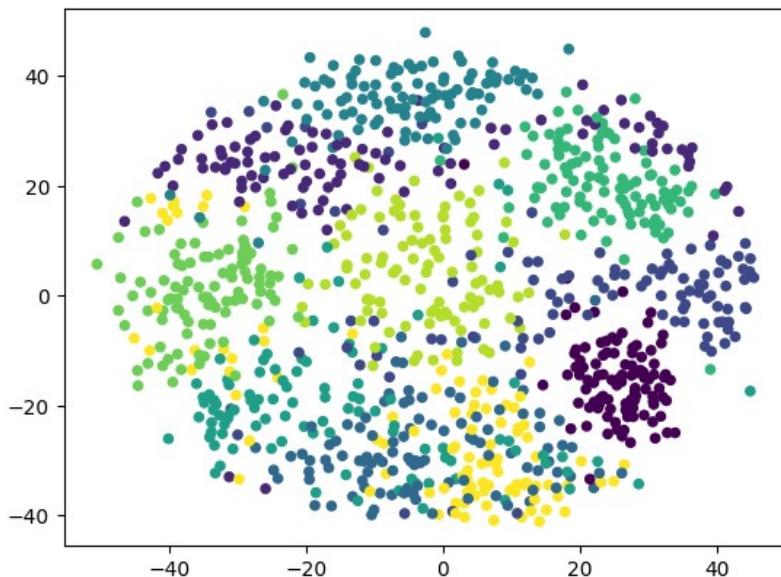


Рис.58. Применение метода MDS к рукописным цифрам. Цвет означает цифру.

Видно, что по сравнению с NMF классы разделились лучше: если в NMF они выглядели как перемешанные разреженные облака, то в MDS они стали более компактными и более разделимыми.

### 9.2.2.2. Стохастическая кодировка соседей(SNE)

Более быстрым является метод SNE (стохастическая кодировка соседей). Его идея другая: если взять точку и посмотреть расстояния до её соседей, то они распределяются особым характерным образом. Можно потребовать чтобы в новом пространстве расстояния от этой точки до соседей были распределены подобным образом. Мы не следим за расстоянием от пробной точки до каждого соседа, мы только требуем, чтобы расстояния до всех соседей были распределены похоже в обоих пространствах. Это сильно ускоряет расчеты. Формально это означает, что при преобразовании координат мы сохраняем не сами

расстояния, а их статистические распределения наблюдаемые из каждой конкретной точки. Отсюда и название "стохастическая" (вероятностная) кодировка.

Математически это формулируется так. Распределения из каждой точки до ее соседей полагаются гауссовыми:

$$\rho_x(x_j|x_i) = \frac{\exp(-\|x_i - x_j\|^2/(2\sigma_{x,i}^2))}{\sum_{k \neq i} \exp(-\|x_i - x_k\|^2/(2\sigma_{x,i}^2))}$$

$$\rho_x(x_i|x_i) = 0$$

$$\rho_y(y_j|y_i) = \frac{\exp(-\|y_i - y_j\|^2/(2\sigma_{y,i}^2))}{\sum_{k \neq i} \exp(-\|y_i - y_k\|^2/(2\sigma_{y,i}^2))}$$

$$\rho_y(y_i|y_i) = 0$$

здесь  $\|\Delta x\|$  - евклидова метрика.

Предполагается, что расстояния в новом пространстве должны быть не равны, а пропорциональны расстояниям в исходном пространстве:

$$\sigma_{x,i} = \alpha \sigma_{y,i}$$

где  $\alpha$  - некая постоянная, единая для всех точек.

Это требование эквивалентно тому, что расстояния могут масштабироваться, но оставаться пропорциональными в обоих системах координат.

При поиске распределений минимизируется дивергенция Кульбака-Лейблера (это некое расстояние между двумя распределениями):

$$KL = \sum_{i,j} \rho_x(x_i|x_j) \log \left( \frac{\rho_x(x_i|x_j)}{\rho_y(y_i|y_j)} \right) \rightarrow \min$$

Минимизацией дивергенции Кульбака-Лейблера удается найти  $\sigma_{x,i}$ , и сгенерировать  $y_i$ .

#### Код 102. Преобразование данных методом SNE

```
from scipy.spatial.distance import pdist, squareform
def sne(X, n_components=2, perplexity=25, max_iter=1000, learning_rate=100.0, momentum=0.5):
    n = X.shape[0]
    D = squareform(pdist(X, 'squared_euclidean'))
    D = np.maximum(D, 1e-12) # Avoid zeros
    P = compute_prob_matrix(D, perplexity)
    P = np.maximum(P, 1e-12)
    Y = np.random.randn(n, n_components) * 1e-4
    Y -= Y.mean(axis=0) # Center
    Y_momentum = np.zeros_like(Y)
    min_error = float('inf')
    for iter in range(max_iter):
        dist_Y = squareform(pdist(Y, 'squared_euclidean'))
        dist_Y = np.maximum(dist_Y, 1e-12)
        Q = np.exp(-dist_Y)
        np.fill_diagonal(Q, 0)
        Q /= Q.sum()
        Q = np.maximum(Q, 1e-12)
        grads = np.zeros_like(Y)
        for i in range(n):
```

```

dY = Y[i] - Y
grads[i] = np.sum((P[i] - Q[i])[:, np.newaxis] * dY, axis=0)
Y_momentum = momentum * Y_momentum - learning_rate * grads
Y += Y_momentum
if iter % 100 == 0:
    error = np.sum(P * np.log(P / Q)) if np.all(Q > 0) else float('inf')
    if error < min_error: min_error = error
    else: break
return Y

def compute_prob_matrix(D, perplexity, tol=1e-5, max_iter=50):
    n = D.shape[0]
    P = np.zeros((n, n))
    for i in range(n):
        beta,beta_min,beta_max = 1.0, -np.inf, np.inf
        distances,log_perp = D[i, :], np.log(perplexity)
        for _ in range(max_iter):
            p_row = np.exp(-beta * distances)
            p_row[i] = 0
            sum_p = p_row.sum()
            if sum_p == 0:
                p_row = np.zeros(n)
                break
            p_row /= sum_p
            entropy = -np.sum(p_row * np.log(p_row + 1e-12))
            current_perplexity = np.exp(entropy)
            if np.abs(current_perplexity - perplexity) < tol: break
            if current_perplexity > perplexity:
                beta_min = beta
                if np.isinf(beta_max): beta *= 2
                else: beta = (beta + beta_max) / 2
            else:
                beta_max = beta
                if np.isinf(beta_min): beta /= 2
                else: beta = (beta + beta_min) / 2
            P[i] = p_row
    P = (P + P.T) / (2 * n)
    return P

# Run SNE
Y = sne(digits.data[:1000], perplexity=25, max_iter=5000, learning_rate=20.0)
plt.scatter(Y[:,0],Y[:,1],c=digits.target[:1000],s=10);

```

Видно, что точки уже начали разделяться в группы, каждую из которых можно интерпретировать как отдельную цифру и в будущем разделить эти группы классификатором или кластеризатором.

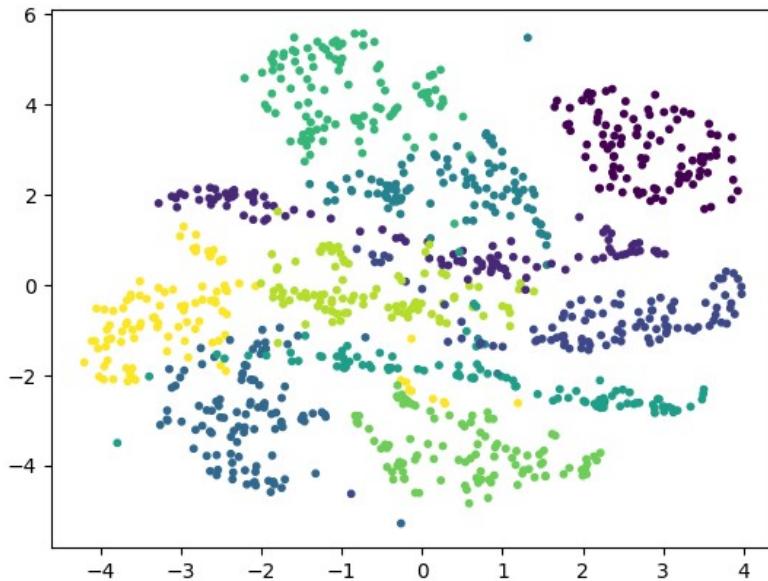


Рис.59. Применение метода SNE к рукописным цифрам. Цвет означает цифру.

### 9.2.2.3. Стохастическая кодировка соседей по $t$ -распределению( $t$ -SNE)

Более современный и эффективный метод — t-SNE (Стохастическая кодировка соседей по  $t$ -распределению). Его отличие в том, что вместо гауссова распределения расстояний для новых координат  $\rho_y$  используется  $t$ -распределение Стьюдента (обычно с одной или двумя степенями свободы):

$$\rho_y(y_j|y_i) = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq i} (1 + \|y_i - y_k\|^2)^{-1}}$$

$$\rho_y(y_i|y_i) = 0$$

здесь  $\|\Delta x\|$  — евклидова метрика. Все остальные части алгоритма совпадают с алгоритмом SNE.

Зачем это нужно? Гауссово распределение имеет "лёгкие хвосты" — малые вероятности найти точку на большом расстоянии, а  $t$ -распределение — более "тяжёлые хвосты" и более резкий пик в нуле, то есть оно плавнее и допускает больше выбросов на малых и больших расстояниях. Поэтому в новом пространстве точек на больших и на малых расстояниях будет немного больше, чем в исходном. А точек на средних расстояниях — немного меньше.

Этот метод позволяет немного "растянуть" точки, расположенные далеко друг от друга, в то время как близко расположенные точки позволяет сблизить. Таким образом, после преобразования t-SNE расстояния между точками внутри одного кластера уменьшаются, а расстояния между точками из разных кластеров — увеличивается.

Это помогает лучше разделять кластеры и визуализировать данные.

Код 103. Преобразование данных методом t-SNE

```
from sklearn.manifold import TSNE
tsne = TSNE(n_components=2)
yp=tsne.fit_transform(digits.data)
plt.scatter(yp[:,0],yp[:,1],c=digits.target,s=10);
```

Видно, что тенденция точек разделяться в группы, которую показала SNE, усилилась и практически все группы явно разделились.

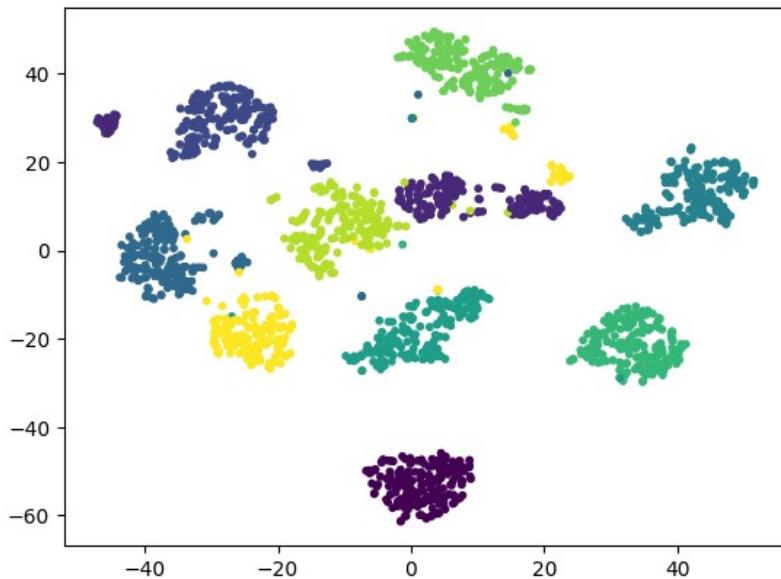


Рис.60. Применение метода t-SNE к рукописным цифрам. Цвет означает цифру.

Теперь мы можем:

- Применить классификатор (например, SVM) или кластеризатор (например GM) для разделения гиперплоскостями
- Сопоставить каждый кластер определенной цифре
- В дальнейшем использовать этот кластеризатор/классификатор для классификации новых изображений

#### 9.2.2.4.Аппроксимация и проекция равномерных многообразий (UMAP)

Везде ранее мы считали, что наше пространство параметров евклидово, и расстояние между двумя точками считается по многомерной теореме Пифагора. На самом деле в большинстве случаев это не так. Это означает, что расстояния по теореме Пифагора можно считать только для малых расстояний, а для больших нужно использовать более сложные формулы.

Метод UMAP (аппроксимация и проекция равномерных многообразий) предполагает, что пространство обладает кривизной (точки расположены не в обычном евклидовом пространстве, а в неком искривленном многомерном пространстве), и приблизить его евклидовым (и вычислять расстояния по теореме Пифагора) можно только локально, в очень малой области. Такое предположение позволяет корректнее разделять данные на кластера, эффективно уменьшая их размеры по сравнению с расстоянием между ними. Более усложненным вариантом этого подхода является например метод РаСМАР.

Посмотрим на его реализацию.

Код 104. Преобразование данных методом UMAP

```
from umap.umap_ import UMAP
```

```

um = UMAP(n_components=2)
yp=um.fit_transform(digits.data[:1000])
plt.scatter(yp[:,0],yp[:,1],c=digits.target[:1000],s=10);

```

Видно, что тенденция точек разделяться в группы, которую показала t-SNE, еще более усилилась (кластера стали меньше, а расстояние между ними - больше), хотя кластера стали приобретать уже несферическую, а вытянутую форму. Получившиеся классы могут быть легко разделены.

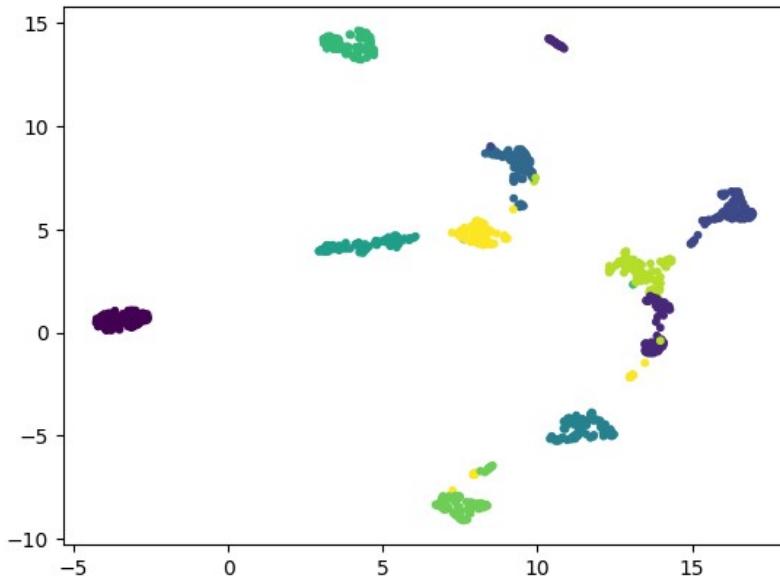


Рис61. Применение метода UMAP к рукописным цифрам. Цвет означает цифру.

### 9.2.3.Полиномиальное спрямляющее пространство

Иногда встречаются случаи, когда размерность пространства признаков мала, а задача сложная. В этом случае, чтобы улучшить качество анализа и использовать более простую модель, имеет смысл повысить размерность пространства.

При анализе закономерностей и построении эффективных признаков одним из важных элементов является спрямляющее пространство. Если целевая функция зависит от признаков  $x, y$  нелинейно, например

$$f = 5x + 10x^2 + 8\sin(y) + 3e^y$$

то имеет смысл ввести новые, нелинейные по  $x, y$  переменные, от которых  $y$  будет зависеть линейно. Такие переменные обычно добавляются к исходным переменным, и делается попытка решать задачу в пространстве более высокой размерности. Например для данного случая от пространства  $X = (x, y)$  мы переходим к пространству  $Y = (x, y, x^2, \sin(y), e^y)$ . В этом пространстве функциональная зависимость  $f$  будет выглядеть намного проще:

$$f = 5Y_0 + 10Y_2 + 8Y_3 + 3Y_4$$

Для такой функции можно использовать более простые методы и модели — линейные, и получившееся пространство называется спрямляющим. Существует теорема Ковера, которая говорит что в пространстве достаточно большой размерности все точки датасета заданного количества линейно разделимы.

Одним из методов построения такого спрямляющего пространства является метод

полиномиальных признаков. Для исходного пространства  $X$  преобразование в полиномиальное пространство степени  $d$  создает новые признаки  $Y$ :

$$y_j = \prod_{i=0}^N x_i^{p_{i,j}}$$

$$\sum_{i=0}^N p_{i,j} \leq d$$

Размерность полиномиального спрямляющего пространства растет очень быстро, как  $\dim(Y) \approx \dim(X)^d$ , поэтому чаще всего используется квадратичное пространство  $d=2$ . В этом случае к обычным компонентам  $X$  добавляются квадраты и перекрестные произведения всех компонент.

Продемонстрировать это можно на таком примере:

Код 105. Повышение размерности пространства методом полиномиального спрямляющего пространства степени 2

```
from sklearn.preprocessing import PolynomialFeatures
x=np.random.randn(5000,3)*2
x[2*x[:,0]**2-x[:,1]>0,2]=1
x[2*x[:,0]**2-x[:,1]<=0,2]=0
pf=PolynomialFeatures(degree=2)
Y=pf.fit_transform(x[:,2:])
fig,axs=plt.subplots(1,2,figsize=(6,3))
axs[0].scatter(x[:,0],x[:,1],c=x[:,2],s=3)
axs[0].set_xlim(-2,2), axs[0].set_ylim(-1,4)
axs[0].set_xlabel('$x_0$'), axs[0].set_ylabel('$x_1$')
axs[1].scatter(Y[:,2],Y[:,3],c=x[:,2],s=3)
axs[1].set_xlabel('$Y_2$'), axs[1].set_ylabel('$Y_3$')
axs[1].set_xlim(-2,5),axs[1].set_ylim(-0.2,5);
```

Из рисунка видно, что при переходе в полиномиальное спрямляющее пространство в новых координатах датасет становится линейно разделимым.

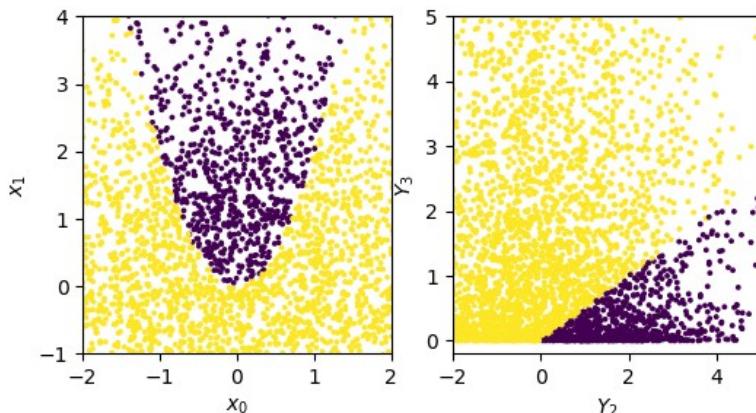


Рис.62. Слева — пример датасета в исходном двумерном пространстве, справа - его-же вид в полиномиальном спрямляющем пространстве

#### 9.2.4.Автоэнкодеры.

Выбирать спрямляющее пространство (и нелинейные преобразования координат)

можно произвольным образом. Но задачу можно сформулировать другим образом: существуют ли некоторые переменные такие, что задав их, можно получить весь набор входных параметров, причем количество этих переменных меньше количества исходных параметров? Такая задача называется задачей поиска латентного (скрытого) представления данных.

Формально она записывается таким образом:

$$\vec{Y} = \vec{F}(\vec{X})$$

$$\vec{X} \approx \vec{F}^{-1}(\vec{Y}) = \vec{F}^{-1}(\vec{F}(\vec{X}))$$

$$\dim(\vec{Y}) < \dim(\vec{X})$$

Обычно задача может быть решена в технологии нейронных сетей поиском оптимального автоэнкодера. Автоэнкодер — это нейронная сеть, составленная из двух блоков, включенных последовательно. Первый блок аппроксимирует функцию  $\vec{F}$ , а второй — функцию  $\vec{F}^{-1}$ . Приведем пример построения автоэнкодера.

#### Код 105. Преобразование данных автоэнкодером

```
import tensorflow as tf
from tensorflow.keras.layers import Input, Dense
from tensorflow.keras.models import Model
x=digits.data/digits.data.max()
input_dim,encoding_dim=x.shape[1],2
input_layer = Input(shape=(input_dim,))
encoder_layer = Dense(input_dim//2, activation='selu')(input_layer)
encoder_layer = Dense(input_dim//4, activation='selu')(encoder_layer)
encoder_layer = Dense(input_dim//8, activation='selu')(encoder_layer)
encoder_layer2 = Dense(encoding_dim, activation='selu')(encoder_layer)
encoder_model = Model(input_layer, encoder_layer2, name='encoder')
decoder_input = Input(shape=(encoding_dim,))
decoder_layer = Dense(input_dim//8, activation='selu')(decoder_input)
decoder_layer = Dense(input_dim//4, activation='selu')(decoder_layer)
decoder_layer = Dense(input_dim//2, activation='selu')(decoder_layer)
decoder_layer2 = Dense(input_dim, activation='sigmoid')(decoder_layer)
decoder_model = Model(decoder_input, decoder_layer2, name='decoder')
autoencoder_input = Input(shape=(input_dim,))
encoded = encoder_model(autoencoder_input)
decoded = decoder_model(encoded)
autoencoder = Model(autoencoder_input, decoded, name='autoencoder')
autoencoder.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-2), loss='binary_crossentropy')
es=tf.keras.callbacks.EarlyStopping(patience=10,monitor='val_loss',mode='min',\
    restore_best_weights=True)
autoencoder.fit(x, x, epochs=10000,batch_size=256,shuffle=True,validation_split=0.2,callbacks=[es])
Yp = encoder_model.predict(x)
```

Из рисунка видно, что некоторые группы цифр оформились в кластера автоэнкодер может эффективно извлекать признаки из самого датасета (при достаточной сложности самого автоэнкодера и достаточно большом объеме датасета для его обучения).

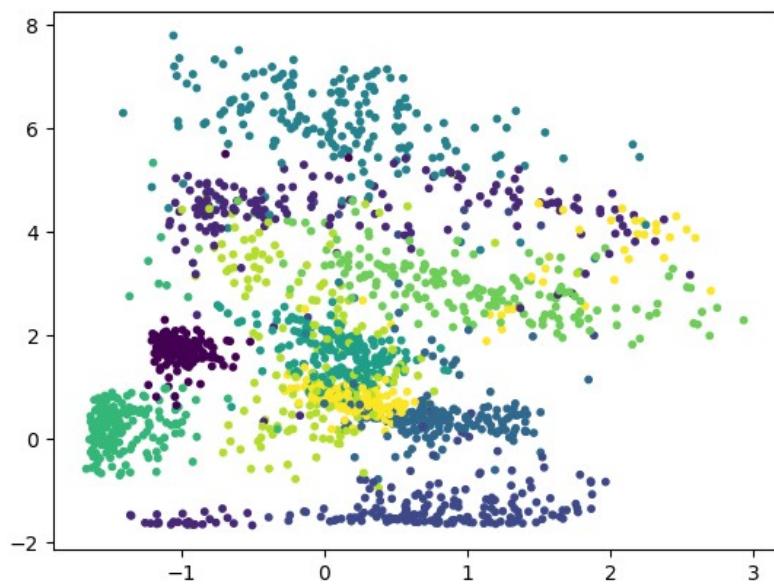


Рис.63. Применение автоэнкодера к рукописным цифрам. Цвет означает цифру.

# 10. Аномалии

Аномалии (или выбросы) — это редкие, нестандартные события или наблюдения, которые значительно отличаются от основного массива данных. Их анализ играет ключевую роль во многих задачах, где помогает выявлять редкие события, нехарактерные для большей части данных.

Аномалии можно разделить на три типа:

1. Глобальные аномалии (выбросы);
2. Контекстные аномалии;
3. Коллективные аномалии.

Почти все аномалии определяются относительно модели. Если данные не соответствуют ожидаемому поведению, они считаются аномальными. Выбор модели критически влияет на результат.

При работе с упорядоченными данными (например, временными рядами) могут встречаться все три типа аномалий в данных.

При работе с неупорядоченными данными контекстные аномалии обычно не встречаются в данных, и мы ищем только глобальные и коллективные аномалии.

## 10.1. Поиск аномалий

### 10.1.1. Глобальные аномалии

Представим упорядоченное множество данных - например, временной ряд измерений температуры на улице. Если внезапно измеряемая температура повысится на 50 градусов - это глобальная аномалия. Такое отклонение может означать либо неисправность термометра, либо пожар. Глобальная аномалия - это выброс, не соответствующий привычным статистическим моделям.

Сгенерируем такой датасет.

Код 106. Тестовый датасет

```
t=np.linspace(0,10,1000)
x=np.sin(t*3)+np.random.randn(1000)*0.1
x[42]+=100
x[420]+=50
x[920]+=30
```

#### 10.1.1.1. Статистические тесты

Если знать распределение точек в датасете, то можно отличать нормальные отчки от аномальных используя доверительные интервалы. Алгоритм действий прост:

1. Строим распределение для заданного датасета;
2. Задаем пороговую вероятность (уровень значимости, например, 1%);
3. Точки, выпадающие из доверительного интервала с заданным уровнем значимости, считаем аномалиями.

При уменьшении уровня значимости мы будем отсекать все более удаленные от центра точки. Один из простейших способов — аппроксимировать распределение нормальным. Рассмотрим случай нормального 95%-го доверительного интервала

### Код 107. Поиск выбросов статистическими методами

```
m,s=x.mean(),x.std()  
y=np.zeros_like(x)  
y[x>m+2*s]=-1;  
y[x<m-2*s]=-1  
plt.scatter(t,x,c=y,s=10)
```

Из рисунка видно, что выбросы выделились достаточно устойчиво.

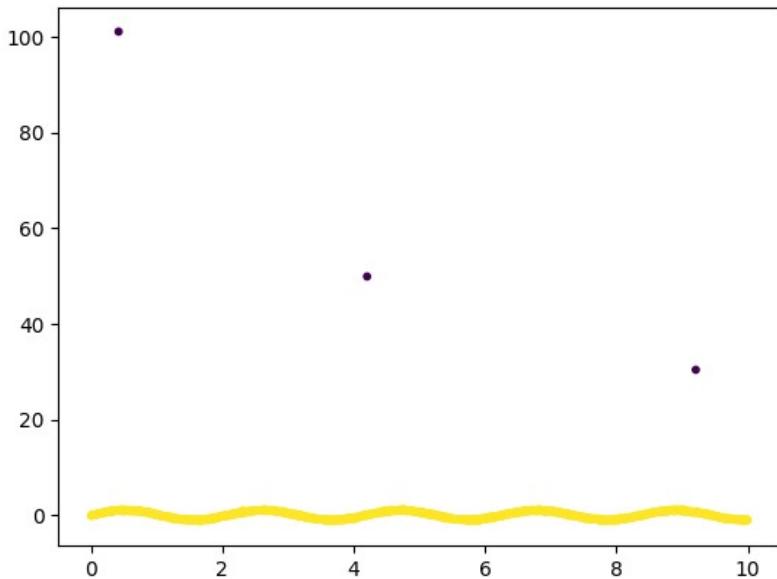


Рис.64. Поиск выбросов статистическими методами

#### 10.1.1.2.Метод интервалов. Ящик с усами.

Рассмотрим случай, когда данные объединены в категории, и нам необходимо найти категорию, существенно отличающуюся от других. Для иллюстрации выберем датасет Boston Housing — статистическую информацию стоимости квартир в Бостоне в зависимости от параметров квартиры и района.

### Код 108. Тестовый датасет — Boston Housing

```
from pandas import read_csv  
from sklearn.model_selection import train_test_split  
url = 'https://raw.githubusercontent.com/jbrownlee/Datasets/master/housing.csv'  
df = read_csv(url, header=None)  
df.head()
```

Метод интервалов — качественный метод, позволяющий выделить аномальные группы по заданному признаку. Принцип заключается в поиске группы (аномалии), доверительный интервал которой не пересекается с доверительными интервалами остальных групп. При визуальном анализе часто используется в подходе «ящик с усами», где усы — это доверительный интервал.

Рассмотрим применение ящика с усами к задаче поиска категорий квартир с аномальной ценой.

Код 109. Поиск аномальных категорий с помощью ящика с усами

```
import seaborn as sns  
sns.barplot(x = 8, y = 13, data = df, capsize=.2)  
plt.show()
```

Из рисунка видно, что категория 24 класса явно выделяется из остальных категорий, и иллюстрирует уменьшение стоимости в зависимости от расстояния до метро.

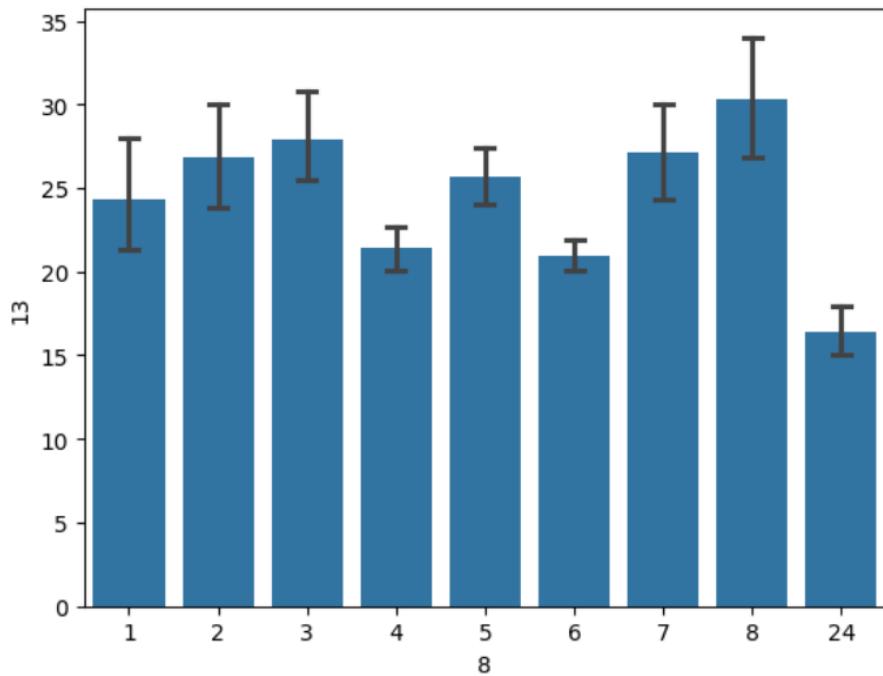


Рис.65. Поиск аномальных категорий с помощью ящика с усами. Зависимость стоимости жилья в зависимости от расстояния до метро. Датасет «Boston Housing».

### 10.1.1.3. Изолирующий лес

Другой моделью является изолирующий лес. Принцип работы изолирующего леса заключается в вычислении средней глубины дерева, на которой прогнозируемая точка выделяется из датасета при случайных разделениях по случайным параметрам. Чем меньше эта глубина, тем более вероятно эта точка является выбросом.

Код 109. Поиск аномальных категорий с помощью изоляционного леса

```
from sklearn.ensemble import IsolationForest  
data = df.values  
X, y = data[:, :-1], data[:, -1]  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=1)  
iso = IsolationForest(contamination=0.05)  
yp = iso.fit_predict(X_test)  
plt.scatter(X_test[:,8],y_test,c=yp), plt.xlabel('Distance'), plt.ylabel('Price');
```

Из рисунка видно, что категория 24 класса выделяется из остальных категорий менее явно.

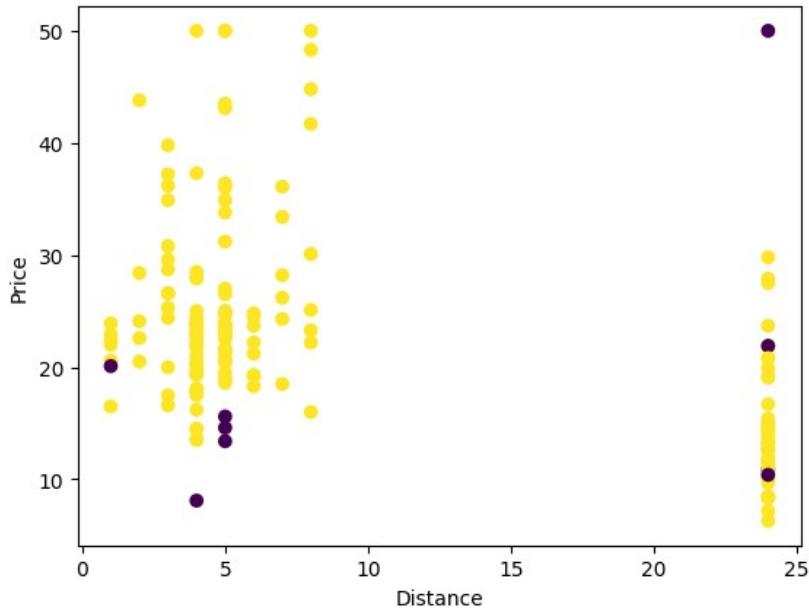


Рис.66. Поиск аномальных категорий с помощью изолирующего леса. Зависимость стоимости жилья в зависимости от расстояния до метро. Датасет «Boston Housing».

#### 10.1.1.4. Одноклассовый метод опорных векторов

Теперь рассмотрим одноклассовый метод опорных векторов (One-Class SVM) — метод для обнаружения аномалий. Это аналог метода опорных векторов (SVM), но для одного класса. Его задача формулируется следующим образом: нам нужно провести разделяющую гиперплоскость так, чтобы "хорошие" точки оказались как можно ближе к началу координат.

One-Class SVM строит границу принятия решений вокруг обычных данных, минимизируя расстояние до начала координат в пространстве признаков, содержащее все обычные точки. Все, что лежит за этой границей, считается аномалией.

Есть два способа обучения:

1. Обучать только на "хороших" данных (без аномалий) — тогда граница ищется для всех точек датасета;
2. Обучать на данных с аномалиями, но их доля не превышает заданного процента (параметр `nu`) - тогда граница ищется так, чтобы заданный процент был за границей, а остальные данные (большинство) — внутри нее.

Проверим работу One-Class SVM на примере.

Код 110. Поиск аномальных категорий с помощью One Class SVM

```
from sklearn.svm import OneClassSVM  
yp = OneClassSVM(gamma='auto',kernel='linear',nu=0.95).fit_predict(X_test)  
plt.scatter(X_test[:,8],y_test,c=-yp), plt.xlabel('Distance'), plt.ylabel('Price');
```

Из рисунка видно, что категория 24 класса выделяется из остальных категорий более уверенно и более часто.

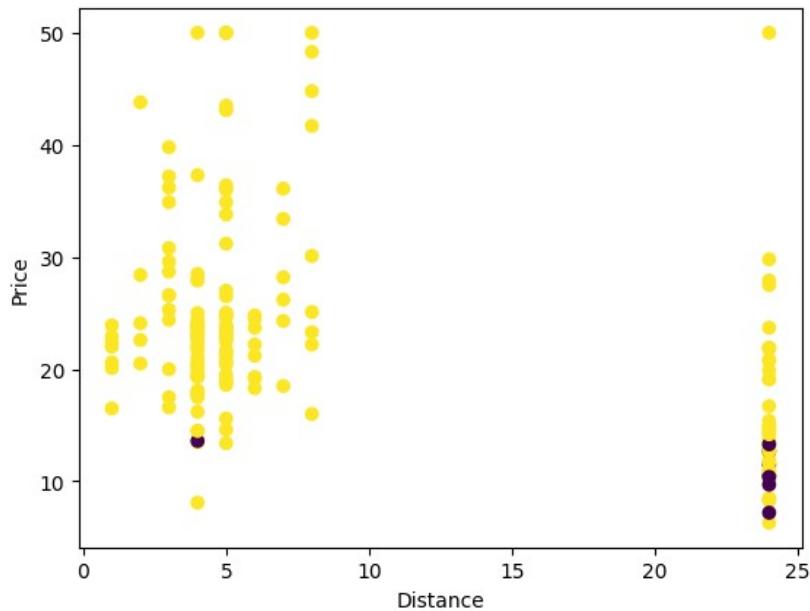


Рис.67. Поиск аномальных категорий с помощью One-class SVM. Зависимость стоимости жилья в зависимости от расстояния до метро. Датасет «Boston Housing».

Плюсом One-Class SVM является гибкость за счет возможности выбора ядра и возможность настройки под конкретные данные. К минусам можно отнести чувствительность к переобучению и требование, чтобы обучающие данные содержали минимум аномалий.

#### 10.1.1.5.Эллиптические методы

Эллиптические методы заключаются в построении эллипсоида в пространстве параметров, описывающего большинство точек. Представляет собой нечто среднее между One-Class SVM и статистическими методами.

Рассмотрим пример

Код 111. Поиск аномальных категорий с помощью эллиптического метода

```
from sklearn.covariance import EllipticEnvelope  
ee = EllipticEnvelope(contamination=0.05)  
yp=ee.fit_predict(X_train)  
plt.scatter(X_train[:,8],y_train,c=yp,s=(-yp+2)*20)
```

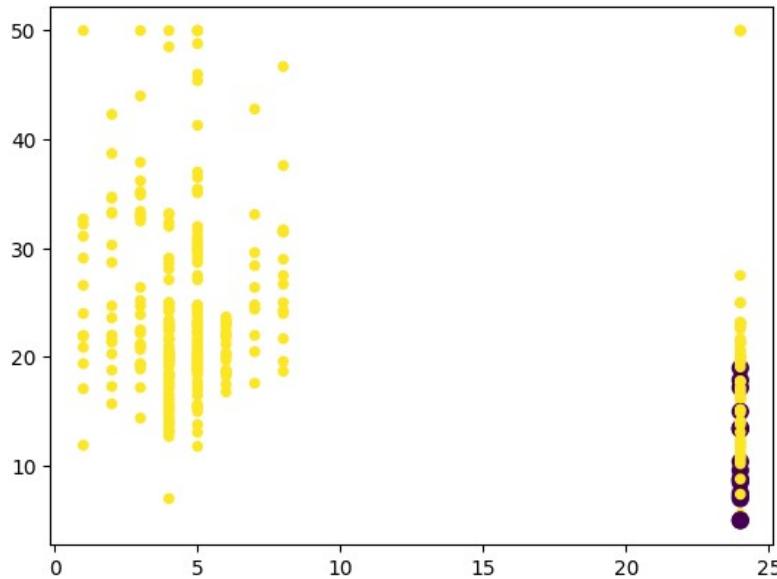


Рис.68. Поиск аномальных категорий с помощью эллиптического метода. Зависимость стоимости жилья в зависимости от расстояния до метро. Датасет «Boston Housing».

#### **10.1.1.6.Локальный уровень выброса, метрические кластеризаторы**

Локальный уровень выброса — это среднее расстояние до нескольких ближайших соседей. Используется предположение что выбросы обычно достаточно далеко расположены от нормальных данных, которых большинство.

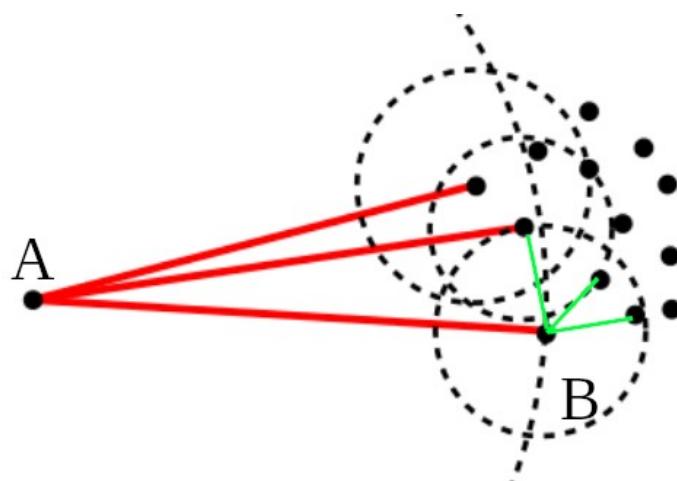


Рис.69. Локальный уровень выброса. Точка А — выброс, точка В — регулярное значение.

Рассмотрим пример его работы.

Код 112. Поиск аномальных категорий с помощью локального уровня выброса
---

```
from sklearn.neighbors import LocalOutlierFactor
lof = LocalOutlierFactor(novelty=False)
yp = lof.fit_predict(X_train)
```

```
plt.scatter(X_train[:,8],y_train,c=yp,s=(-yp+2)*20)
```

Видно, что для этого датасета метод работает хуже.

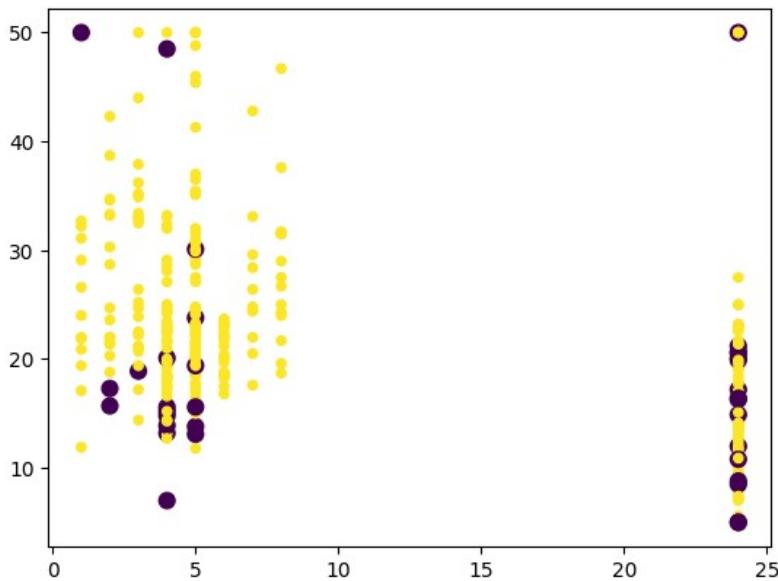


Рис.70. Поиск аномальных категорий с помощью локального уровня выброса. Зависимость стоимости жилья в зависимости от расстояния до метро. Датасет «Boston Housing».

Аналогично локальному уровню выброса могут работать и метрические методы кластеризации, например алгоритм DBSCAN, размечающий все изолированные точки отдельным классом (-1). Изолированными он считает точки, выделившиеся в классы с малым количеством элементов (этот режим в реализации DBSCAN библиотеки sklearn контролируется параметром `min_samples`)

### 10.1.2. Контекстные аномалии

Теперь представим, что мы измеряем температуру за окном. Обычно ночью холоднее, днем теплее из-за Солнца. Если ночью температура вдруг поднимется до дневных значений - это контекстная аномалия. Она не выходит за общие пределы значений, но нарушает привычную динамику. Такой выброс можно заметить только при учете временного контекста.

Контекстная аномалия отличается от глобальной тем, что глобальный выброс значительно превышает любые вариации модели, а контекстный находится в пределах привычных статистических вариаций. Если бы мы не зависимость данных от упорядоченной переменной, например времени, контекстную аномалию можно было бы не заметить - она бы выглядела как обычное значение в общем распределении.

Для выявления контекстных аномалий необходимо иметь модель прогноза поведения анализируемого параметра по упорядоченной переменной (например по времени). Самая простая модель — линии Боллинджера — скользящий статистический метод, выход значения за пределы доверительного интервала по скользящему окну:

```
Код 113. Поиск контекстных аномалий с помощью линий Боллинджера
```

```

def detect_ca(data, window_size=25):
    anomalies = np.zeros_like(data)
    for i in range(0,data.shape[0] - window_size,1):
        window = data[i:i + window_size]
        q1 = np.percentile(window, 5)
        q3 = np.percentile(window, 95)
        iqd=q3-q1
        an=0
        if data[i+window_size]<q1-1.5*iqd: an=1
        if data[i+window_size]>q3+1.5*iqd: an=1
        anomalies[i+window_size]=an
    return anomalies

import numpy as np
import matplotlib.pyplot as plt
t=np.linspace(0,10,1000)
x=np.sin(t*3)+np.random.randn(1000)*0.1
x[142]+=1
x[400]+=1
x[820]+=1
yp=detect_ca(x,window_size=10)
plt.scatter(range(x.shape[0]), x, c=-yp, s=(yp+1)*20);

```

Результат по тестовому датасету приведен на рисунке:

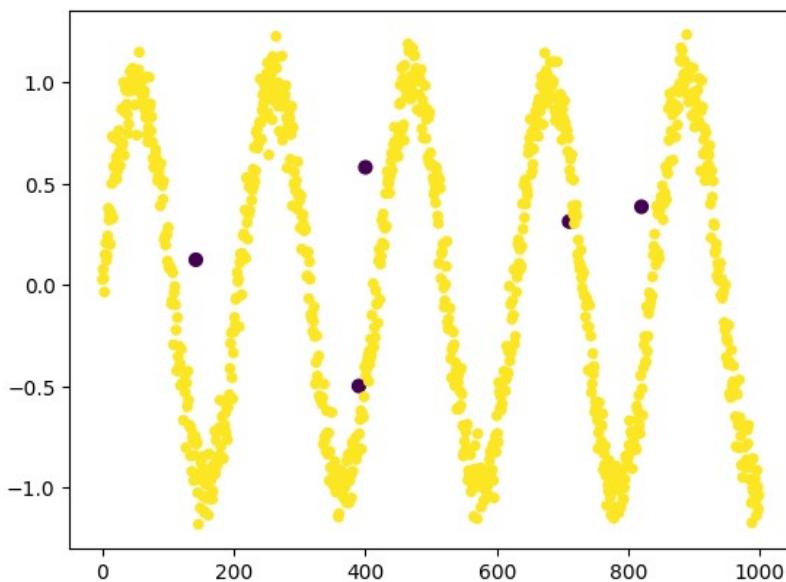


Рис.71. Поиск аномальных значений с помощью линий Боллинджера.

### 10.1.3. Коллективные аномалии

Коллективные аномалии - это самый сложный для обнаружения тип. Коллективная аномалия связана с изменением соотношений между различными координатами векторов данных. Например, если в наборе данных обычно выполняется соотношение

$x_0 \approx x_1 + x_2 + x_3 + 2x_4$ , но в нескольких точках эта зависимость резко нарушается - это коллективная аномалия.

Коллективные аномалии — это группа точек данных, которая в целом отличается от остального набора из-за соотношений между параметрами или из-за их коллективного поведения, хотя отдельные параметры могут казаться нормальными. Обнаружить такие аномалии сложнее всего, потому что по отдельности все переменные могут выглядеть привычно, проблема только в нарушении их взаимосвязи.

Обычно задача решается построением модели для данных и оценке — укладываются данные в модель или нет.

Рассмотрим пример оценки качества электрокардиограммы по ее форме. Это классическая задача на коллективные аномалии и связана с формой кардиограммы, а не с абсолютными значениями ее точек. Воспользуемся для этого построением автоэнкодера с размерностью внутреннего представления данных 8:

#### Код 114. Поиск коллективных аномалий в электрокардиограммах с помощью автоэнкодера

```
import pandas as pd
import tensorflow as tf
from sklearn.model_selection import train_test_split
dataframe=pd.read_csv('http://storage.googleapis.com/download.tensorflow.org/data/ecg.csv',\
                      header=None)
raw_data = dataframe.values
labels,data = raw_data[:, -1], raw_data[:, 0:-1]
train_data, test_data, train_labels, test_labels = train_test_split(data, labels)
min_val,max_val = tf.reduce_min(train_data), tf.reduce_max(train_data)
train_data = (train_data - min_val) / (max_val - min_val)
test_data = (test_data - min_val) / (max_val - min_val)
train_data,test_data = tf.cast(train_data, tf.float32),tf.cast(test_data, tf.float32)
train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)
normal_train_data,normal_test_data = train_data[train_labels],test_data[test_labels]
anomalous_train_data,anomalous_test_data = train_data[~train_labels],test_data[~test_labels]
autoencoder=tf.keras.models.Sequential([
    tf.keras.layers.Dense(32, activation="relu"), tf.keras.layers.Dense(16, activation="relu"),
    tf.keras.layers.Dense(8, activation="relu"), tf.keras.layers.Dense(16, activation="relu"),
    tf.keras.layers.Dense(32, activation="relu"), tf.keras.layers.Dense(140, activation="sigmoid")
])
autoencoder.compile(optimizer='adam', loss='mae')
es=tf.keras.callbacks.EarlyStopping(patience=3)
autoencoder.fit(normal_train_data, normal_train_data,
                epochs=1000,batch_size=512,validation_split=0.2,callbacks=[es])
train_loss1 = tf.keras.losses.mae(autoencoder.predict(normal_train_data), normal_train_data)
train_loss2 = tf.keras.losses.mae(autoencoder.predict(anomalous_test_data), anomalous_test_data)
meant=np.mean(train_loss)
stdt=np.std(train_loss)
threshold=meant+1.5*stdt
train_loss=list(train_loss1)+list(train_loss2)
yp=(np.array(train_loss)>threshold).astype(int)
labels=[0]*train_loss1.shape[0]+[1]*train_loss2.shape[0]
from sklearn.metrics import classification_report
print(classification_report(labels,yp))
```

Этот автоэнкодер обучается исключительно на регулярных данных, после чего по

качеству восстановления этих данных определяется порог, выше которого результат считается аномальным. Итоговый отчет о классификации, выполненный как по нормальным, так и по аномальным данным показывает качество прогноза  $F_1=0.9$ :

	precision	recall	f1-score	support
0	0.99	0.92	0.95	2215
1	0.75	0.98	0.85	546
accuracy			0.93	2761
macro avg	0.87	0.95	0.90	2761
weighted avg	0.95	0.93	0.93	2761

Рис.72. Качество прогноза аномальных ЭКГ автоэнкодером, обученным на нормальных кардиограммах.

## 10.2. Построение моделей, устойчивых к аномалиям в обучающих данных

В большей части случаев задача исследователя состоит не в поиске аномалий, а создании модели для данных содержащих аномалии. Очевидно, что удаление аномалий — только один из способов построить эффективную модель. Другим способом является построение модели, устойчивой к аномалиям в обучающем датасете. Такие модели могут выступать как фильтрами аномалий, так и их детекторами в случае коллективных и контекстных аномалий. Обычно в качестве таких моделей используются уже известные нам автоэнкодеры.

Наиболее распространенным подходом к этой задаче является обучение нейронной сети. Существует много способов построения моделей, устойчивых к аномалиям: аугментация данных, и использование функций потерь, устойчивых к аномалиям.

При обучении нейронных сетей важной величиной является функция потерь. Многие функции потерь устойчивы к аномальным выбросам. Одна из наиболее часто используемых в таких случаях функция потерь — MAE.

Аугментация данных позволяет при обучении автоэнкодера добавлять шум на входе требуя восстановить по зашумленным данным оригинал. Такая модель тоже становится устойчивой к искажениям того типа, который использовался при аугментации.

## **11.Анализ оттока**

Отток (Churn) - это явление, когда клиенты прекращают пользоваться продуктом или услугой компании. Цель прогнозирования оттока — заранее выявлять клиентов с высокой вероятностью ухода, чтобы успеть принять меры по их удержанию.

Отток может проявляться в разных формах:

Явный отток - клиент официально расторгает договор (например, отключает подписку);

Неявный отток - клиент перестает быть активным (не заходит в приложение, не совершает покупки).

Способы решения задачи оттока:

1. Когортный анализ - разделение данных на группы (когорты) по какому-либо категориальному признаку и выделение групп, выделяющихся по уровню оттока. Обычно применяется для оценки вероятности

2. Модели машинного обучения – предсказание для каждого сэмпла вероятности оттока.

Многие задачи могут быть сведены к прогнозу оттока. Обычно это прогноз вероятности для бинарной целевой переменной по небольшому обучающему датасету с очень большим количеством признаков.

### **11.1.Задача оттока с категориальными переменными**

Преимущества категориальных переменных в возможности перекодировки и перестановки их значений. Анализ оттока клиентов строится по следующему принципу.

Этап 1. Определение типов переменных - числовые или категориальные переменные.

Метод разделения - для каждой переменной считаем количество уникальных значений и анализируем их число. Если оно меньше некого порога – считаем переменную категориальной.

Этап 2. Если категориальных признаков очень много – выявление наиболее сильно коррелирующих с целевой бинарной переменной (проще всего корреляцией V-Крамера), остальные игнорируются.

Этап 3. Построение распределения средних значений оттока по каждому значению каждой категориальной переменной. Оценка вероятности (p-value) что среднее значение по группе отличается от среднего значения оттока по всему датасету. Решается например Т-критерием для средних.

Этап 4. Корректировка полученных p-value на множественность гипотез. Чаще всего по Бенджамини-Хохбергу.

Этап 5. Интерпретация полученных категорий и их значений.

При необходимости можно оценить вероятность оттока любого нового элемента, рассчитав полную вероятность его оттока как произведение соответствующих вероятностей оттока по каждой из категорий-значений к которым он принадлежит.

### **11.2.Задача оттока с числовыми переменными**

Проще всего решить эту задачу переводом числовых переменных в категориальные. Для этого используется дискретизация. Дискретизация бывает двух основных типов – равномерная и квантильная. При равномерной дискреты значений имеют одинаковую ширину, при квантильной дискреты выбирают так, чтобы в каждый дискрет попадало примерно поровну элементов датасета.

После дискретизации задача сводится к задаче с категориальными переменными.

## 12. Тематическое моделирование

Задача тематического моделирования возникает во многих областях, где объект описывается суперпозицией своих различных состояний (их вероятностями), и требуется категоризация (кластеризация) таких объектов. Тематическое моделирование можно назвать кластеризацией статистических распределений.

Наиболее широко метод тематического моделирования применяется при анализе текстов. Суть задачи заключается в том, чтобы, взглянув на текст, кратко определить, к какой области он относится. Это связано с выявлением ключевых слов в определенной сфере и категоризацией текстов по тематикам. Такое моделирование помогает оптимизировать например поиск в поисковых системах.

Аналогичная задача возникает в проблеме прогноза оценки товара по профилю пользователя, известной как прогнозирование рекомендаций. Если вместо товаров по горизонтали и пользователей по вертикали мы возьмём слова и документы, то получим аналогичную структуру. Мы можем разбить фильмы на группы, характерные для определённых пользователей. Например, одна группа пользователей предпочитает боевики, другая — ужастики. Это приводит к тому, что фильмы, относящиеся к боевикам, будут выделяться для одних пользователей, а фильмы ужасов — для других.

Давайте рассмотрим задачу подробнее. Представим матрицу встречаемости слов в различных текстах. По горизонтали у нас расположены номера документов, а по вертикали — слова. Чем темнее область, тем чаще слово встречается в соответствующем тексте.

Формально, у нас есть матрица, где в ячейках стоят числовые значения (неотрицательные). Эта матрица обладает следующим свойством: мы можем переставлять строки и столбцы этой матрицы без потери смысла, так как порядок фильмов и пользователей не важен. Наша задача — перестановкой строк и столбцов собрать вместе вблизи диагонали блоки, в которых значения максимальны. Такие блоки называются темами (топиками).



Рис.73. Пример перемешивания матриц для приведения к виду, используемому для решения задачи. Слева – исходная матрица, справа – матрица после перестановки.

В рамках задачи классификации текстов это сводится к стандартной задаче классификации по входному вектору — матрице эмбеддингов. А что делать, если мы не знаем, на какие классы мы должны разделить документы? Такая задача встречается при анализе тематик текста — интернетовские, журнальные или газетные статьи обычно обладают какой-то тематикой, но сложно конкретизировать эту тематику заранее или заранее оценить их число. Поэтому для таких задач имеет смысл построить схему, которая будет эти

тематики определять самостоятельно: мы не будем знать название тематики, но для нас главное - чтобы система как-то разделила документы на классы, а уж потом мы их проинтерпретируем и назовем. Задача разделения набора текстов на заранее неизвестные классы называется тематическим моделированием. Ее гиперпараметром является число тем.

Тематическое моделирование — способ построения модели коллекции текстовых документов, которая определяет, к каким вероятным темам относится каждый из документов. Тематическая модель определяет, к каким темам относится каждый документ и какие слова образуют каждую тему.

Детально можно посмотреть работу [9].

## 12.1.Латентно-семантический анализ (LSA)

Существует много разных способов построить тематическую модель. Один из самых простых - это метод латентно-семантического анализа (LSA).

При анализе текстов обычно используют матрицу с неотрицательными значениями, например TF-IDF (частота термина, делённая на частоту документа) или его логарифмическую версию. После вычислений получается представление, где наша матрица (описывающая слова и документы) выражается через произведение трех матриц.

$$M = U \Sigma V$$

Первая матрица ( $U$ ) имеет слова по вертикали и некие параметры по горизонтали. Вторая матрица ( $V$ ) содержит номера документов по вертикали и некие параметры по горизонтали. При перемножении этих матриц получается исходная матрица "слова × документы".

Эти неизвестные параметры в матрицах  $U, V$  мы интерпретируем как темы. Веса в матрицах показывают важность тем. Например, первое слово может часто встречаться в теме 1 и редко в темах 2 и 3, а второе слово - наоборот. Эти скрытые (латентные) параметры мы и называем темами.

У этого произведения будет понятная интерпретация:

- первая матрица ( $U$ ) говорит о том, с какими весами какие слова у вас встречаются в темах;
- размерность диагональной матрицы  $\Sigma$  - это количество тем и их встречаемость (вес);
- последняя матрица  $V$  говорит о том, к каким темам у нас наиболее вероятно относится документы.

Таким образом, основная задача тематического моделирования - представить матрицу весов в виде произведения трех матриц, одна из которых будет отражать соотношение между словами и скрытыми темами, другая – веса тем в этом корпусе текстов, а третья – соотношение между темами и словами.

Проиллюстрируем метод укороченным SVD на 3 компоненты:

Код 115. Тематическое моделирование методом LSA

```
M=np.array([[0.2,0,0,0],
           [0.1,0.9,0,0],
           [0.1,0,0,0],
           [0,0,0.4,0.3],
           [0.6,0.1,0.6,0.7]])
from numpy.linalg import svd
U,S,V=svd(M,full_matrices=False)
print(M)
print(np.round(U,1)[:5,:3])
print(np.round(np.diag(S[:3]),1),1)
print(np.round(V,1)[:3,:4])
```

```
print(np.round(U[:5,:3]@np.diag(S[:3])@V[:3,:4],1))
```

Итоговый вид представления LSA-представления матрицы M будет выглядеть, как:

$$\begin{bmatrix} 0.2 & 0. & 0. & 0. \\ 0.1 & 0.9 & 0. & 0. \\ 0.1 & 0. & 0. & 0. \\ 0. & 0. & 0.4 & 0.3 \\ 0.6 & 0.1 & 0.6 & 0.7 \end{bmatrix} = \begin{bmatrix} -0.1 & 0. & -0.5 \\ -0.2 & -1. & 0.1 \\ -0. & 0. & -0.3 \\ -0.3 & 0.2 & 0.8 \\ -0.9 & 0.2 & -0.2 \end{bmatrix} \times \begin{bmatrix} 1.2 & 0. & 0. & 0. \\ 0. & 0.9 & 0. & 0. \\ 0. & 0. & 0.3 & 0. \\ -0.3 & 0.2 & 0.8 & 0. \end{bmatrix} \times \begin{bmatrix} -0.5 & -0.2 & -0.6 \\ 0. & -1. & 0.2 \\ -0.8 & 0.1 & 0.5 \\ -0.2 & 0. & -0.6 \end{bmatrix}$$

Рис.74. Итоговый вид представления LSA.

В результате метода главных компонент (сингулярного разложения) получается квадратная диагональная матрица  $\Sigma$  с убывающими положительными сингулярными значениями на диагонали. Кроме неё появляются две дополнительные поворотные матрицы - U и V. Размерность матрицы тем, а соответственно и векторных представлений, можно уменьшить рассмотренными ранее способами, ограничив возможные значения сингулярных значений  $\lambda_i$  либо по абсолютному значению, либо пользуясь различными внутренними критериями, обсуждавшимися ранее.

Проблема интерпретации этого метода - отрицательные веса в матрицах U,V: он не идеален для тематического моделирования, так как допускает их отрицательные значения, что ухудшает интерпретируемость результатов.

Также у этого алгоритма есть недостатки, связанные с эффективной численной реализацией: при добавлении нового документа d в коллекцию, матрицу V (распределение  $p(t|d)$ ) невозможно вычислить по тем же формулам, что и для предыдущей коллекции, и нужно перестраивать всю модель заново.

## 12.2.Вероятностный латентно-семантический анализ (PLSA)

Более продвинутыми моделью это является вероятностный латентно-семантический анализ (PLSA). Уже известной нам альтернативой методу РСА является неотрицательное матричное разложение (NMF).

$$M = U V$$

Оно разлагает матрицу M на произведение двух неотрицательных матриц U и V. Поэтому она имеет более понятную интерпретацию (все веса  $\geq 0$ ) и сохраняет структуру: слова x темы (U) и документы x темы (V). Остаётся вопрос приоритезации: как определить главную тему для слова, если оно встречается в нескольких темах с разными весами? Решение - нормировать столбцы матриц U,V (сумма весов по каждой теме = 1), что позволяет сравнивать вклады разных тем. Поэтому мы нормируем каждый столбец так, чтобы сумма его элементов равнялась единице. Это преобразование соответствует переходу от числовых значений к вероятностям, где все вероятности неотрицательны и сумма вероятностей всегда равна 1.

Таким образом мы переходим к вероятностной постановке задачи. В этом случае у нас получается мягкая классификация - мы можем говорить о вероятностях принадлежности. Когда оба столбца (слова x темы и документы x темы) нормированы и имеют смысл вероятностей, интерпретация становится проще. Нормировка позволяет сравнивать вклады разных тем, устранивая проблему разных масштабов признаков.

При таком подходе каждый элемент матричного произведения можно интерпретировать как распределение вероятности. Неотрицательное матричное разложение

(NMF) приводит к виду, где:

$$P(w, d) = \sum_t P_t(t) P_d(d|t) P_w(w|t) = P_d(d) \sum_t P_t(t|d) P_w(w|t)$$

Это выражение аналогично формуле полной вероятности:  $P_w(w|d)$  вычисляется по статистике слов в документах, а  $P_w(w|t)$  и  $P_t(t|d)$  - неизвестные распределения, которые нужно оценить.

Вероятностная модель появления пары «документ-слово» может быть записана тремя эквивалентными способами:

$$\begin{aligned} p(d, w) &= \sum_{t \in T} p(w|t) p(t) p(d|t) \\ &= \sum_{t \in T} p(w|t) p(t|d) p(d) \\ &= \sum_{t \in T} p(d|t) p(t|w) p(w) \end{aligned}$$

Суммирование идет по темам  $t$ , здесь  $T$  - множество тем.

Здесь  $p_t(t)$  - неизвестное априорное распределение тем во всей коллекции;  $p_d(d)$  - априорное распределение на множестве документов, его эмпирическая оценка;  $p_d(d) = n_d/n$ , где  $n = \sum n_d$  - суммарная длина всех документов,  $n_d$  - длина документа  $d$ ;  $p_w(w)$  - априорное распределение на множестве слов, его эмпирическая оценка;  $p_w(w) = n_w/n$ , где  $n_w$  - число вхождений слова  $w$  во все документы.

Задача моделирования методом PLSA - найти неизвестные условные распределения  $p_w(w|t)$ ,  $p_t(t|d)$ . С качественной точки зрения, эта задача соответствует задаче нахождения матриц  $U, V$  в методе LSA.

В рамках этого подхода задача тематического моделирования решается в вероятностной постановке методом максимального правдоподобия. Основное предположение: вероятность появления слова в документе определяется только тематикой документа. Задача сводится к минимизации функционала максимального правдоподобия для матриц  $P(\text{слово}| \text{тема})$  и  $P(\text{тема}| \text{документ})$ , на которые мы разлагаем исходную матрицу  $P(\text{слово}| \text{документ})$ .

Такая постановка задачи представляет собой особый вид неотрицательного матричного разложения (NMF), где матрицы не только неотрицательны, но и нормированы по строкам (или столбцам).

Решение всегда численное и итеративное, вычисляемое с определенной точностью. Модельные распределения полагаются мультиномиальными:

$$\begin{aligned} P(\alpha_0, \alpha_1, \dots, \alpha_k | p_0, p_1, \dots, p_k) &= \frac{n!}{\alpha_0! \dots \alpha_k!} p_0^{\alpha_0} p_1^{\alpha_1} \dots p_k^{\alpha_k} \\ \sum_i \alpha_i &= n \end{aligned}$$

где искомые параметры – вероятности объектов (слов, документов, тем):  $p_0, p_1, \dots, p_k$ .

Мультиномиальное распределение вероятностей аналогично бросанию многогранного кубика: если биномиальное распределение описывает многоократное бросание монеты (с 2 исходами) и наблюдение за комбинациями исходов (числом выпадений орла), то мультиномиальное распределение описывает многоократное бросание кубика (с множеством граней) и наблюдение за комбинациями исходов (комбинациями выпавших очков).

Чаще всего для поиска параметров распределения используют EM-алгоритм: на E-шаге вычисляются вспомогательные параметры, на M-шаге по оптимальным значениям этих параметров вычисляются коэффициенты матриц. Эти шаги чередуются до достижения нужной точности приближения.

Рассмотрим пример работы алгоритма PLSA на нашем тестовом датасете:

## Код 116. Тематическое моделирование методом PLSA

```
def plsa(X, n_topics, max_iter=100, tol=1e-6, random_state=42):
    X = np.array(X, dtype=np.float64)
    n_docs, n_words = X.shape
    P_z = np.random.rand(n_topics)
    P_z /= P_z.sum()
    P_d_given_z = np.random.rand(n_topics, n_docs)
    P_d_given_z /= P_d_given_z.sum(axis=1, keepdims=True)
    P_w_given_z = np.random.rand(n_topics, n_words)
    P_w_given_z /= P_w_given_z.sum(axis=1, keepdims=True)
    P_z_given_d_w = np.zeros((n_docs, n_words, n_topics))
    prev_log_likelihood = -np.inf
    for iteration in range(max_iter):
        for d in range(n_docs): # --- E-step: Compute P(z | d, w) ---
            for w in range(n_words):
                if X[d, w] > 0:
                    numerator = P_z * P_d_given_z[:, d] * P_w_given_z[:, w]
                    denom = numerator.sum()
                    if denom > 0: P_z_given_d_w[d, w, :] = numerator / denom
                    else: P_z_given_d_w[d, w, :] = 1.0 / n_topics # uniform fallback
        P_z_new = np.zeros(n_topics) # --- M-step: Update P(z), P(d|z), P(w|z) ---
        P_d_given_z_new = np.zeros((n_topics, n_docs))
        P_w_given_z_new = np.zeros((n_topics, n_words))
        for d in range(n_docs):
            for w in range(n_words):
                if X[d, w] > 0:
                    count = X[d, w]
                    for z in range(n_topics):
                        gamma = count * P_z_given_d_w[d, w, z]
                        P_z_new[z] += gamma
                        P_d_given_z_new[z, d] += gamma
                        P_w_given_z_new[z, w] += gamma
        if P_z_new.sum() > 0: P_z = P_z_new / P_z_new.sum()
        else: P_z = np.ones(n_topics) / n_topics
        for z in range(n_topics):
            if P_d_given_z_new[z].sum() > 0: P_d_given_z[z] = P_d_given_z_new[z] / \
                P_d_given_z_new[z].sum()
            else: P_d_given_z[z] = np.ones(n_docs) / n_docs
        for z in range(n_topics):
            if P_w_given_z_new[z].sum() > 0: P_w_given_z[z] = P_w_given_z_new[z] / \
                P_w_given_z_new[z].sum()
            else: P_w_given_z[z] = np.ones(n_words) / n_words
        log_likelihood = 0.0
        for d in range(n_docs):
            for w in range(n_words):
                if X[d, w] > 0:
                    pw = np.sum(P_z * P_d_given_z[:, d] * P_w_given_z[:, w])
                    if pw > 1e-10: log_likelihood += X[d, w] * np.log(pw)
```

```

else: log_likelihood += X[d, w] * np.log(1e-10) # avoid log(0)
if abs(log_likelihood - prev_log_likelihood) < tol: break
prev_log_likelihood = log_likelihood
return P_d_given_z.T, P_w_given_z, P_z

doc_topic_matrix, word_topic_matrix, Pz = plsa(M, n_topics=3)
print(np.round(doc_topic_matrix,2))
print(np.round(word_topic_matrix,2))
print(np.round(Pz,2))
res=doc_topic_matrix@np.diag(Pz)@word_topic_matrix
print(np.round(res/res.sum(),2))
print(np.round(M/M.sum(),2))

```

Результат работы алгоритма:

$$\begin{bmatrix} [0.2 & 0. & 0.] & [0. & 0. & 0.2] & [[0.5 & 0. & 0.] & [0.01 & 0.99 & 0. & 0.] & [0.05 & 0. & 0. & 0.] \\ [0.1 & 0.9 & 0.] & [0.9 & 0. & 0.09] & X & [0. & 0.27 & 0.] & X & [0. & 0. & 0.5 & 0.5] & = & [0.03 & 0.22 & 0. & 0.] \\ [0.1 & 0. & 0.] & \approx & [0. & 0. & 0.1] & [0. & 0. & 0.23] & [1. & 0. & 0. & 0.] & [0.02 & 0. & 0. & 0.] \\ [0. & 0. & 0.4 & 0.3] & [0. & 0.35 & 0. ] & & & & [0. & 0. & 0.09 & 0.09] \\ [0.6 & 0.1 & 0.6 & 0.7] & [0.1 & 0.65 & 0.61] & & & & [0.15 & 0.02 & 0.16 & 0.16] \end{bmatrix}$$

Рис.75. Исходная матрица, результат разложения методом PLSA и оптимальное произведение UV

## 12.3.Латентное размещение Дирихле (LDA)

Более широко используемым на настоящее время является тематический анализ с использованием латентного размещения Дирихле (LDA). Оно эквивалентно латентному семантическому анализу, но вид неизвестных распределений фиксирован - они представляет собой параметризованные распределения Дирихле:

$$P(p_1, p_2, \dots, p_K | \alpha_1, \alpha_2, \dots, \alpha_K) = \frac{1}{B(\vec{\alpha})} \prod_{i=1}^K p_i^{\alpha_i - 1} = Dir(\vec{p}, \vec{\alpha})$$

здесь  $p_i$  – вектора (вероятности) документов или вектора тем в зависимости от использования в задаче. Эти векторные представления нормированы в сумме на единицу (поскольку это вероятности):

$$\sum_{i=1}^K p_i = 1$$

Здесь  $\alpha_i$  – неизвестные (скрытые) параметры распределения,  $B(\vec{\alpha})$  – некая нормировочная функция (многомерная бета-функция). Для распределения документы-темы обычно вводится параметры  $\vec{\alpha}$ , для распределения темы-слова –  $\vec{\beta}$ .

Из сравнения мультиномиального распределения и распределения Дирихле видно, что они похожи, и основное отличие в параметрах и гиперпараметрах. Для мультиномиального распределения гиперпараметрами являются вероятности  $\vec{p}$  и исследуется зависимость от параметров – показателей степеней  $\vec{\alpha}$ , с которыми эти вероятности входят в распределение. Для распределения Дирихле гиперпараметрами являются показатели степеней  $\vec{\alpha}$ , и исследуется зависимость от параметров – вероятностей  $\vec{p}$ , которые входят в это размещение с этими степенями. При этом в мультиномиальном распределении коэффициенты  $\vec{\alpha}$  целые неотрицательные числа в сумме равные заданному числу, а в распределении Дирихле — произвольные действительные числа. В мультиномиальном распределении вероятности  $\vec{p}$

произвольны, а в распределении Дирихле — в сумме равны единице. В статистике распределение Дирихле называют сопряженным априорным распределением к мультиномиальному (апостериорному) распределению.

Преимущество распределения Дирихле в его универсальности - варьируя коэффициенты  $\vec{\alpha}, \vec{\beta}$ , можно получить практически любое распределение: от равномерного до резко выраженного (пикообразного). Это позволяет гибко настраивать форму распределения частот в документах и темах. Другим преимуществом является его генеративный характер — возможность работы с новыми данными.

Ключевое преимущество LDA - все коэффициенты всегда остаются неотрицательными и могут интерпретироваться как вероятности и модель может работать с новыми данными без дообучения. Результаты аналогичны SVD (те же ключевые слова в темах), но с вероятностной интерпретацией.

Таким образом, в нашей задаче есть некие скрытые параметры  $\vec{\alpha}, \vec{\beta}$  и количество этих скрытых параметров пропорционально количеству тем.

Таким образом, задача тематического моделирования сводится к задаче определения этих неизвестных параметров распределений Дирихле, чтобы можно было исходную вероятность появления слов в документах представить в виде произведения распределений условных вероятностей – слов в темах и тем в документах.

Вероятностная модель появления пары «документ-слово» в LDA может быть записана в виде:

$$p(d, w) = \sum_{t \in T} p_w(w|t) p_d(t|d)$$

Суммирование идет по темам Т.

При этом, вектора документов  $p_d(t|d)$  порождаются одним и тем же вероятностным распределением на нормированных Т-мерных векторах; это распределение является параметрическим распределением Дирихле  $Dir(\vec{p}_d, \vec{\alpha})$ , а вектора слов  $p_w(w|t)$  порождаются одним и тем же вероятностным распределением на нормированных W-мерных векторах; это распределение является параметрическим распределением Дирихле  $Dir(\vec{p}_w, \vec{\beta})$ .

#### Код 117. Тематическое моделирование методом LDA

```
from sklearn.decomposition import LatentDirichletAllocation
lda = LatentDirichletAllocation(n_components=3, evaluate_every=1)
doc_topic_matrix = lda.fit_transform(M/M.sum(axis=1, keepdims=True))
Comp=lda.components_ / lda.components_.sum(axis=1)[:, np.newaxis]
print(M)
print(np.round(doc_topic_matrix, 2))
print(np.round(Comp, 2))
out=(doc_topic_matrix)@(Comp)
print(np.round(out/out.sum(axis=1, keepdims=True), 1))
print(np.round(M/M.sum(axis=1, keepdims=True), 1))
```

Результат работы алгоритма при  $\alpha_i=1/3, \beta_i=1/3$ :

$$\begin{bmatrix}
 [1. & 0. & 0. & 0.] & [0.17 & 0.17 & 0.66] & [0.16 & 0.54 & 0.15 & 0.15] & [0.5 & 0.2 & 0.2 & 0.1] \\
 [0.1 & 0.9 & 0. & 0.] \approx [0.63 & 0.17 & 0.2] & \times [0.12 & 0.12 & 0.4 & 0.37] = [0.3 & 0.4 & 0.2 & 0.2] \\
 [1. & 0. & 0. & 0.] & [0.17 & 0.17 & 0.66] & [0.72 & 0.09 & 0.09 & 0.09] & [0.5 & 0.2 & 0.2 & 0.1] \\
 [0. & 0. & 0.6 & 0.4] & [0.17 & 0.66 & 0.17] & & [0.2 & 0.2 & 0.3 & 0.3] \\
 [0.3 & 0.1 & 0.3 & 0.4] & [0.19 & 0.5 & 0.31] & & [0.3 & 0.2 & 0.3 & 0.2]
 \end{bmatrix}$$

Рис.76. Исходная матрица (нормированная), результат разложения методом LDA и оптимальное произведение UV (нормированное)

При работе с текстами можно использовать gensim.models.LdaModel, ориентированную на обработку массивов текстов.

## 12.4.LDA с регуляризациями.BigARTM

У NMF/PLSA/LDA есть проблема - решение не единственное. Чтобы сузить множество решений, обычно добавляют регуляризирующие слагаемые, используя метод множителей Лагранжа. Это преобразует исходную задачу в задачу с дополнительным ограничивающим членом - некоторой функцией от матриц.

Если разложение матриц имеет вид:

$$P(w, d | \vec{\alpha}, \vec{\beta}) = \sum_{t \in T} \phi_{w,t} \theta_{t,d}$$

то решение задачи разложения методом максимального правдоподобия имеет вид:

$$\left\{
 \begin{array}{l}
 \sum_{d \in D} \sum_{w \in W} \left( n_{d,w} \ln \left( \sum_{t \in T} \phi_{w,t} \theta_{t,d} \right) \right) + R(\Phi, \Theta) \rightarrow \max_{\Phi, \Theta} \\
 \sum_{w \in W} \phi_{w,t} = 1, \phi_{w,t} \geq 0 \\
 \sum_{t \in T} \theta_{t,d} = 1, \theta_{t,d} \geq 0
 \end{array}
 \right.$$

здесь  $R(\Phi, \Theta)$  - регуляризующее слагаемое, наиболее часто вида:

$$R(\Phi, \Theta) = \sum_{t, w} (\beta_w - 1) \ln \phi_{w,t} + \sum_{d, t} (\alpha_t - 1) \ln \theta_{t,d}$$

Проблема регуляризации (выбора оптимального решения из множества возможных) решается через специальные регуляризующие слагаемые, позволяющие выбрать наиболее подходящий вариант. В качестве регуляризующего слагаемого выступают специальные функции с двумя векторами параметров:

1.  $\vec{\beta}$  (разный для каждого слова, зависит от параметра  $w$ ), регуляризирует распределение  $\phi_{w,t}$  слов по темам
2.  $\vec{\alpha}$  (разный для каждой темы), регуляризирует распределение  $\theta_{w,t}$  документов по темам

Рассмотрим возможные варианты регуляризации и их влияние на результаты.

Первый подход - разделение на предметные и фоновые темы. Часто встречается ситуация, когда среди тематических статей (например, в физической энциклопедии статьи про электричество, атмосферу или океан) присутствуют служебные материалы - введение, заключение, содержание. В таких текстах используются слова, не относящиеся к основной тематике (например, благодарности редакторам или информация об авторах).

Для выделения таких фоновых тем можно использовать специальную регуляризацию. Если задать большие положительные значения параметра  $\beta$  для определенных тем (например, для 5 фоновых тем), распределение слов в них станет практически равномерным. Для предметных тем мы устанавливаем малые значения  $\beta$ , что приводит к резко выраженному распределению с несколькими ключевыми словами. Целью такой

регуляризации – разделение тем на фоновые темы (большие  $\beta$ , равномерное распределение) и предметные темы (малые  $\beta$ , пикообразное распределение).

Второй подход - выделение лексического ядра. Идеальное тематическое разбиение должно минимизировать пересечение между темами (области на диаграмме должны максимально не перекрываться). Для этого мы можем:

- Уменьшать размеры пересекающихся тем
- Исключать дублирующиеся темы
- Увеличивать количество уникальных слов в каждой теме

Фактически это означает необходимость минимизировать корреляцию между словами в разных темах - чтобы одни и те же слова реже встречались одновременно в нескольких темах. Для этого можно использовать специальный регуляризующий множитель, представляющий собой произведение элементов матрицы  $\phi_{w,t}$  (частот слов по темам).

$$R(\Phi) = -\frac{\tau}{2} \sum_{t \in T} \sum_{s \in T/t} \sum_{w \in W} \phi_{w,t} \phi_{w,s} \rightarrow \max$$

здесь  $\tau$  — коэффициент регуляризации

Максимизация этого произведения приводит к "сжатию" тематических облаков - они становятся более компактными и меньше пересекаются между собой.

Аналогичным образом можно бороться с "расщепленными" темами - ситуациями, когда одна тема фактически дублируется в двух вариантах. Изменяя вид регуляризирующего слагаемого, можно накладывать дополнительные условия на распределение слов по темам.

Простейший вариант регуляризации - задание весов тем (первая тема самая важная, вторая менее важна и т.д.). В библиотеке gensim это контролируется параметрами alpha (для матрицы "документ-тема") и eta (для матрицы "слово-тема"), позволяющими управлять важностью документов и слов в темах.

Цель такой регуляризации - достичь максимальной определенности, когда по набору слов можно однозначно определить принадлежность к теме.

Наиболее полно способы регуляризации реализуются в пакете BigARTM. Ее использование позволяет:

- Выделять компактные тематические кластеры;
- Исключать фоновые темы (через отрицательные весовые коэффициенты)
- Получать "острые" темы с ярко выраженными ключевыми словами (увеличивая абсолютное значение отрицательного  $\beta$ )

## **Заключение**

В работе сделан обзор некоторых текущих методов анализа данных с примерами на языке Питон. Рекомендуется читать книгу одновременно с «Введение в Большие данные и машинное обучение (конспект лекций)».

Работа выполнена при финансовой поддержке Фонда Владимира Потанина (грант #ГСГК-065/25)

# **Литература**

1. Маккинли У., Python и анализ данных // М.: ДМК Пресс, 2015.
2. Грас Дж., Data Science. Наука о данных с нуля//ВНВ, 2020, 416с
3. О'Нил К., Шатт Р., Data Science. Инсайдерская информация для новичков. Включая язык R. //Питер, 2019, 368с
4. NIST/SEMATECH e-Handbook of Statistical Methods,  
<http://www.itl.nist.gov/div898/handbook/> <https://doi.org/10.18434/M32189>
5. Бендат Дж., Пирсол А., Прикладной анализ случайных данных//М., Мир, 1989, 540с.
6. Hamilton, J.D. Time Series Analysis// Princeton University Press, 1994, 799с,  
<https://doi.org/10.2307/j.ctv14jx6sm>
7. Залманзон Л.А., Преобразования Фурье, Уолша, Хаара и их применение в управлении связи и других областях//М.,Наука, 1989. 496 с.
8. Ghojogh B., Crowley M., Karray F., Ghodsi A., Elements of Dimensionality Reduction and Manifold Learning //Springer, 2023, 606с, <https://doi.org/10.1007/978-3-031-10602-6>
9. Воронцов К. В. Вероятностное тематическое моделирование: теория регуляризации ARTM и библиотека с открытым кодом BigARTM. 2024.  
<http://www.machinelearning.ru/wiki/images/d/d5/Voron17survey-artm.pdf>

## **Дополнительная литература**

10. Chan S.H., Introduction to Probability for Data Science//Michigan Publishing, 2021, 690с
11. Bruce P., Bruce A., and Gedeck P., Practical Statistics for Data Scientists//O'Reilly, 2020, 342с
12. James G., Witten D., Hastie T., Tibshirani R., Taylor J., An Introduction to Statistical Learning with Applications in Python//Springer, 2023, 607с
13. Nield T., Essential Math for Data Science//O'Reilly, 2022, 332п.
14. Гонсалес Д., Лая С., Нолан Д., Изучаем Data Science: обработка, исследование, визуализация и моделирование данных с помощью Python//ВНВ, 2025, 560с  
(<https://github.com/DS-100/textbook>)

Оформление обложки — Я.О.Бернгардт

DOI: 10.5281/zenodo.16777543