**Конспект Яндекс Учебника по ML**

[Типы задач 2](#_Toc195632960)

[Метрики качества, функции потерь, отличия 3](#_Toc195632961)

[Метод градиентного спуска и стохастического градиентного спуска (СГД) 5](#_Toc195632962)

[Метод наименьших квадратов (МНК) 7](#_Toc195632963)

[Регуляризация 8](#_Toc195632964)

[Метод опорных векторов (SVM) 9](#_Toc195632965)

[Метод K-ближайших соседей (KNN) 10](#_Toc195632966)

[KMeans 11](#_Toc195632967)

[Решающие деревья (Decision Trees) 12](#_Toc195632968)

[Random Forest и градиентный бустинг 13](#_Toc195632969)

[Стэкинг и Бэггинг (Stacking & Bagging) 16](#_Toc195632970)

[Кросс-валидация 19](#_Toc195632971)

[Полносвязные нейронные сети 20](#_Toc195632972)

[Метод обратного распространения ошибки 22](#_Toc195632973)

[Оптимизация при обучении нейронных сетей 24](#_Toc195632974)

[Свёрточные нейронные сети 26](#_Toc195632975)

[Нейросети для работы с последовательностями 28](#_Toc195632976)

[Языковые модели в машинном обучении 30](#_Toc195632977)

[Кластеризация в ML 31](#_Toc195632978)

[Метод главных компонент (PCA) 33](#_Toc195632979)

[Предобработка данных 35](#_Toc195632980)

[Прочее 39](#_Toc195632981)

# Типы задач

**Задачи обучения с учителем – для каждого объекта обучающей выборки известны правильные ответы заранее:**

* Регрессия – предсказание продолжительности поездки на каршеринге, погода в городе на завтра.
* Бинарная классификация – вернет клиент кредит в установленный срок или нет, сдаст ли студент сессию.
* Многоклассовая классификация – определение предметной области научной статьи (математика, биология, информатика и т. д.).
* Многоклассовая классификация с пересекающимися классами – автоматическое проставление тегов для ресторанов.
* Ранжирование – ранжирование поисковой выдачи.

**Отличия регрессии и классификации:**

1. **Тип переменной:**
   * **Регрессия:** Прогнозирование количественных переменных (например, стоимость, цена, вес).
   * **Классификация:** Прогнозирование категориальных переменных (например, виды растений, классы клиентов, типы заболеваний).
2. **Методы и алгоритмы:**
   * **Регрессия:** используются методы линейной регрессии, полиномиальной регрессии и другие.
   * **Классификация:** используются методы логистической регрессии, деревьев решений, метода опорных векторов (SVM), нейронных сетей.

**Задачи обучения без учителя – известны только данные, а ответы неизвестны или вообще не существуют:**

* Кластеризация – разделение документов из электронной библиотеки по темам.

# Метрики качества, функции потерь, отличия

**Определение:** Метрика качества используется для оценки качества модели после или во время ее обучения.

**Цель:** Оценка и сравнение моделей с точки зрения их производительности на валидационных или тестовых данных.

**Влияние на обучение:** Метрика качества не влияет напрямую на процесс обучения, а лишь предоставляет информацию о том, насколько хорошо модель работает на отдельных примерах или целых наборах данных.

**Метрики качества:**

* **Классификация: Accuracy (доля правильных ответов), precision (объекты, классифицированные положительно и действительно положительны), recall (доля объектов положительного класса из всех объектов), F1-score (гармоническое среднее precision и recall), AUC-ROC (площадь ошибок под кривой). False Positive – ошибка 1 рода, False Negative – ошибка второго рода. Confusion Matrix – бинарная классификация.**
* **Регрессия: MSE (а также RMSE и RMPE), MAE (а также MAPE), R2-score (разница между выборками в данных и прогнозами модели)**

**Функция потерь (Loss Function)**

**Определение:** Функция потерь измеряет разницу между предсказанными значениями модели и фактическими значениями для конкретного примера в обучающих данных.

**Цель:** Минимизация функции потерь помогает оптимизировать параметры модели в процессе обучения.

**Влияние на обучение:** Функция потерь напрямую влияет на процесс обучения, так как градиенты этой функции используются для обновления весов в модели.

**Пример:**

* **Для классификации:** Перекрестная энтропия. Она измеряет разницу между предсказанными вероятностями классов и истинными метками классов.
* **Для регрессии:** Среднеквадратичное отклонение. Измеряет разницу между предсказанными значениями и фактическими значениями путем суммирования квадратов разностей.

**Loss — это то, что минимизирует алгоритм обучения (техническая часть).**

**Metric — то, что интересно вам и бизнесу (интерпретируемая оценка).**

# Метод градиентного спуска и стохастического градиентного спуска (СГД)

**Метод градиентного спуска** — это итеративный алгоритм оптимизации, используемый для минимизации функции потерь. Он работает путем вычисления градиента функции потерь относительно параметров модели и обновления этих параметров в направлении, противоположном градиенту. Это позволяет находить значения параметров, которые уменьшают ошибку предсказаний модели.

**Скорость обучения (learning rate)** — это гиперпараметр, который определяет размер шага при обновлении параметров модели. Если скорость обучения слишком мала, обучение будет медленным и может занять много времени. Если же скорость обучения слишком велика, это может привести к пропуску минимума функции потерь и нестабильности в процессе обучения

**Как работает?**

1. Выбирается начальная точка (случайные веса модели).
2. На каждом шаге вычисляется градиент (вектор частных производных) функции потерь.
3. Обновляются веса
4. Процесс повторяется, пока не достигается минимум

**Плюсы (+):**

* Гарантированно сходится к локальному минимуму
* Подходит для гладких выпуклых функций (MSE в линейной регрессии)

**Минусы (-):**

* Медленный на больших данных – надо считать градиент на каждом шаге по всем примерам
* Может застрять в локальном минимуме
* Чувствителен к выбору скорости обучения (слишком большой – расходится, слишком маленький – долго учится)

**Стохастический градиентный спуск (СГД)** – ускоренная версия градиентного спуска, где на каждом шаге градиент вычисляется только по одному случайному примеру (или небольшой подвыборке – **минибатчу**)

**Плюсы (+):**

* Быстрее обычного градиентного спуска, так как не требует полного прохода по данным.
* Лучше уходит от локальных минимумов, за счет случайного выбора параметров
* Подходит для огромных дата-сетов

**Минусы (-):**

* Высокая дисперсия градиентов
* Может не сойтись точно к минимуму
* Требует аккуратного подбора скорости обучения

**Чаще всего используется mini-batch SGD –** градиент считается по небольшой группе примеров (32-256 штук), используется почти во всех современных нейросетях – GPT, BERT...

# Метод наименьших квадратов (МНК)

**Метод наименьших квадратов (МНК) – простейший пример постановки задачи линейной регрессии.**

МНК ищет такую линию, чтобы сумма квадратов ошибок (расстояний от реальных точек до линии) была минимальной. Ошибка — это разница между реальным значением и предсказанным линией.

Минусы: чувствительность к выбросам, предполагает только линейную зависимость, требует выполнение условий Гаусса-Маркова (дисперсия ошибок постоянна, отсутствует автокорреляция, отсутствует мультиколлинеарность, модель линейна), не подходит для категориальных переменных, не учитывает сложные структуры данных, плохо работает при малом количестве данных.

# Регуляризация

Регуляризация — это метод борьбы с переобучением (overfitting), когда модель слишком хорошо запоминает обучающие данные, но плохо работает на новых.

Суть: добавить в функцию потерь "штраф" за сложность модели, чтобы она стала более гладкой и обобщаемой.

Типы регуляризации:

* L1-регуляризация (Lasso) – обнуляет некоторые веса и проводит автоматический отбор признаков, полезно при большом количестве неважных фичей. Делает модель разреженной, помогает в отборе признаков, нестабилен при сильно коррелированных признаках.
* L2-регуляризация (Ridge) – сжимает веса, но не обнуляет их, делает модель более устойчивой к шуму. Стабильнее L1 при мультиколлинеарности, но не отбирает признаки.
* ElasticNet (L1 + L2) – дает баланс между признаками и устойчивостью. Лучше, когда много коррелированных признаков, но сложнее настраивать, так как два гиперпараметра.
* Dropout (в нейросетях) – выключает часть нейронов во время обучения.
* Ранняя остановка – останавливает обучение, когда ошибка на валидации начинает расти.
* Data Augmentation – добавление искусственных данных для разнообразия обучающей выборки

# Метод опорных векторов (SVM)

SVM – алгоритм для классификации и регрессии, который ищет оптимальную разделяющую границу между классами.

**Основная идея**

Цель SVM: найти такую гиперплоскость (линию, плоскость и т.д.), которая:

* Максимально отделяет классы друг от друга.
* Имеет наибольший зазор (margin) до ближайших точек каждого класса.

Опорные векторы — это точки, которые находятся ближе всего к разделяющей границе и определяют её положение.

**Плюсы (+):**

* Эффективен в высокомерных пространствах, даже когда признаков больше, чем объектов
* Работает с нелинейными данными
* Устойчив к переобучению

**Минусы (-):**

* Требует аккуратного подбора параметров
* Медленно работает на больших датасетах
* Плохо интерпретируется

# Метод K-ближайших соседей (KNN)

**KNN (K-Nearest Neighbors, метод k-ближайших соседей)** — это алгоритм классификации и регрессии, основанный на принципе схожести объектов. Является алгоритмом обучения с учителем.

**Основная идея**

* Для нового объекта алгоритм ищет k ближайших (по расстоянию) точек в обучающей выборке.
* Для классификации выбирается класс, который встречается чаще среди соседей.
* Для регрессии возвращается среднее (или медиана) значений соседей.

**Шаги алгоритма**

* Выбор метрики расстояния (обычно Евклидово, но может быть Манхэттен, косинусное и др.).
* Вычисление расстояний от новой точки до всех точек в данных.
* Выбор k ближайших соседей.
* Голосование (для классификации) или усреднение (для регрессии).

**Пример**

Допустим, у нас есть данные о цветах (по длине и ширине лепестков), и мы хотим классифицировать новый цветок:

Если k=3, и среди соседей 2 "Ириса" и 1 "Роза", то новый цветок → "Ирис".

Если k=5, и значения соседей = [5, 6, 7, 5, 6], то прогноз регрессии = среднее (5.8).

# KMeans

**K-Means — это популярный алгоритм кластеризации, который группирует данные в заранее заданное число кластеров (k). Он работает итеративно, минимизируя внутрикластерное расстояние (дисперсию).**

**Как работает K-Means?**

* Инициализация центроидов **-** Выбирается количество кластеров k, случайно или с помощью специальных методов (K-Means++) инициализируются центры кластеров (центроиды).
* Назначение точек кластерам - каждая точка данных приписывается к ближайшему центроиду (обычно по евклидову расстоянию).
* Пересчёт центроидов - центроиды перемещаются в среднее (медианное) положение точек их кластера.
* **Повторение шагов 2–3**

**Процесс продолжается, пока:**

* центроиды перестают значительно меняться,
* кластеры стабилизируются,
* достигнуто максимальное число итераций.

# Решающие деревья (Decision Trees)

Решающие деревья – это алгоритм машинного обучения, который строит иерархическую структуру правил для классификации или регрессии. Разбивает данные на подгруппы, чтобы минимизировать неопределенность.

**Как работает?**

**Основная идея:** Дерево состоит из узлов (вопросы о признаках) и листьев (ответы — классы или численные значения).

**Ключевые этапы построения**

* Выбор лучшего признака для разделения — используется метрика (энтропия, Gini, MSE).
* Разделение данных — создаются подгруппы по условию
* Рекурсия — процесс повторяется для каждой подгруппы, пока не выполнится критерий остановки.

**Критерии разделения:**

* Классификация – энтропия, индекс Джини
* Регрессия – MSE (минимизация дисперсии в подгруппах)

**Плюсы (+):**

* Интерпретируемость – правила видны в виде дерева
* Работает с категориальными и числовыми данными без предобработки
* Не требует масштабирования признаков

**Минусы (-):**

* Склонность к переобучению – без ограничений дерево подстроится под шум.
* Неустойчивость – малые изменения данных могут сильно менять структуру
* Плохо обобщает линейные зависимости

Критерии остановки – максимальная глубина дерева, минимальное число образцов в листе, минимальные прирост информации.

# Random Forest и градиентный бустинг

**Random Forest — это ансамблевый алгоритм, который строит множество решающих деревьев и объединяет их предсказания для повышения точности и устойчивости.**

**Как работает?**

* **Бутстрэп (Bootstrap Aggregating, Bagging)** - из обучающей выборки случайно выбираются подвыборки с повторением (обычно того же размера). Каждое дерево обучается на своей подвыборке.
* **Случайность в признаках -** при каждом разделении узла рассматривается не все признаки, а только случайное подмножество (например, sqrt(n\_features)).
* **Голосование или усреднение -** классификация: большинство голосов деревьев, регрессия: среднее предсказаний.

**Плюсы (+):**

* Устойчив к переобучению
* Работает с пропущенными значениями и неотмасштабированными данными
* Дает оценку важности признаков

**Минусы (-):**

* Менее интерпретируем, чем одно дерево
* Требует больше вычислительных ресурсов

**Градиентный бустинг** — это ансамблевый метод, где деревья строятся последовательно, каждое новое дерево корректирует ошибки предыдущих.

**Как работает?**

* **Инициализация -** начальное предсказание (например, среднее значение для регрессии).
* **Последовательное обучение -** для каждой итерации: вычисляются остатки (разница между предсказанием и истиной), новое дерево обучается предсказывать эти остатки, предсказание обновляется.
* **Регуляризация:** используются параметры learning\_rate, max\_depth, n\_estimators.

**Популярные реализации:**

* **XGBoost (eXtreme Gradient Boosting) –** оптимизация скорости и точности. Деревья строятся по принципу «Последовательно по уровням до достижения максимальной глубины».
* **LightGBM –** работает с большими данными, использует gradient-based одностороннюю выборку (GOSS). Деревья строятся по принципу «На каждом шаге делим вершину с наилучшим скором».
* **CatBoost –** автоматически обрабатывает категориальные признаки. Строит деревья по принципу «Все вершины одного уровня имеют одинаковый предикат».

**Плюсы (+):**

* Высокая точность (выше случайного леса)
* Гибкость – можно выбрать разные функции потерь

**Минусы (-):**

* Склонен к переобучению при неправильных настройках
* Требует тщательной настройки (grid search)

Поскольку для построения градиентного бустинга достаточно уметь считать градиент функции потерь по предсказаниям, с его помощью можно решать широкий спектр задач. В библиотеках градиентного бустинга даже реализована возможность создавать свои функции потерь: для этого достаточно уметь вычислять ее градиент, зная истинные значения и текущие предсказания для элементов обучающей выборки.

Типичный градиентный бустинг имеет в составе несколько тысяч деревьев решений, которые необходимо строить последовательно. Построение решающего дерева на выборках типичного размера и современном железе, даже с учетом всех оптимизаций, требует небольшого, но всё-таки заметного времени (0.1-1c), которое для всего ансамбля превратится в десятки минут. Это не так быстро, как обучение линейных моделей, но всё-таки значительно быстрее, чем обучение типичных нейросетей.

A screenshot of a computer

AI-generated content may be incorrect.

# Стэкинг и Бэггинг (Stacking & Bagging)

Оба метода относятся к ансамблевым алгоритмам, которые комбинируют несколько моделей для улучшения предсказаний. Однако работают они по-разному.

**Бэггинг (Bootstrap Aggregating)**

**Как работает?**

* Бутстрэп-выборки: из обучающих данных случайно создаются N подвыборок с повторением (обычно такого же размера, как исходные данные).
* Параллельное обучение: на каждой подвыборке обучается одна и та же модель (например, дерево решений).
* Агрегация предсказаний: для классификации — голосование большинством; для регрессии — усреднение результатов.

**Примеры алгоритмов**

* Random Forest (множество деревьев с бэггингом + случайные признаки).
* BaggingClassifier в sklearn.

**Плюсы (+):**

* Уменьшает дисперсию (переобучение) особенно для нестабильных моделей (деревьев)
* Параллелизуется – можно обучать модели независимо

**Минусы (-):**

* Не улучшает смещение – если базовая модель слабая, то и ансамбль будет плохим

**Стэкинг (Stacking)**

**Как работает?**

* **Базовые модели (Level-0):** обучаются разные алгоритмы (например, SVM, дерево, логистическая регрессия) на исходных данных.
* **Мета-модель (Level-1):** предсказания базовых моделей становятся новыми признаками для финальной модели (обычно линейной или простой).
* **Кросс-валидация во избежание утечек:** чтобы мета-модель не переобучалась, базовые модели предсказывают out-of-fold (OOF) данные.

**Плюсы (+):**

* Может достигать высокой точности, комбинируя сильные стороны разных моделей
* Гибкость – можно сочетать любые алгоритмы

**Минусы (-):**

* Сложность и риск переобучения
* Требует больше вычислительных ресурсов

A screenshot of a computer

AI-generated content may be incorrect.

# Кросс-валидация

**Зачем нужна кросс-валидация?**

* Оценка качества модели на недоступных данных (тестовом наборе).
* Помогает избежать переобучения и случайных колебаний в оценках.

**Методы кросс-валидации**

* **Hold-Out (отложенная выборка):** данные делятся на train (обучение) и test (валидация) один раз. Проблема: Зависимость от разбиения → высокая дисперсия оценки.
* **K-Fold (K-блочная кросс-валидация):** данные разбиваются на K частей (фолдов). K-1 фолдов — обучение, 1 фолд — валидация. Процесс повторяется K раз. Итоговая оценка: Среднее качество по всем итерациям. Оптимальное K: обычно 5 или 10.
* **Leave-One-Out (LOO):** Частный случай K-Fold, где K = N (N — число объектов). Плюс: максимально точная оценка. Минус: вычислительно дорого (N моделей).
* **Stratified K-Fold:** сохраняет распределение классов в каждом фолде (важно для несбалансированных данных).
* **Time Series CV (для временных рядов):** учитывает порядок данных: обучаются только на прошлых данных, тестируют на "будущих".

**Как выбрать метод?**

* Стандартные данные (без временной зависимости): K-Fold (K=5/10)
* Несбалансированные классы: Stratified K-Fold.
* Мало данных: LOO (но медленно).
* Временные ряды: Time Series Split.

**Где применять?** Подбор гиперпараметров, сравнение моделей.

**Что избегать?** Утечки данных (например, предобработка до разбиения).

# Полносвязные нейронные сети

1. **Что такое полносвязная нейросеть (Fully Connected Network)?**
   * Также называется **многослойный перцептрон (MLP)**.
   * Состоит из последовательности слоёв, где **каждый нейрон связан со всеми нейронами предыдущего слоя**.
2. **Архитектура:**
   * **Входной слой** (количество нейронов = числу признаков).
   * **Скрытые слои** (произвольное количество нейронов).
   * **Выходной слой** (зависит от задачи: 1 нейрон для регрессии, несколько для классификации).

**Обучение нейросети:**

1. **Прямое распространение (Forward Pass):**
   * Данные проходят через все слои, вычисляется предсказание.
2. **Функция потерь (Loss Function):**
   * **Для регрессии:** MSE (Mean Squared Error).
   * **Для классификации:** Cross-Entropy (Кросс-энтропия).
3. **Обратное распространение ошибки (Backpropagation):**
   * Алгоритм **градиентного спуска** корректирует веса, минимизируя ошибку.
   * **Цепное правило** (chain rule) для расчёта градиентов.
4. **Оптимизаторы:**
   * **SGD (Stochastic Gradient Descent)** — базовый метод.
   * **Adam** — адаптивный оптимизатор (рекомендуется для начала).

**Плюсы и минусы полносвязных сетей**

✅ **Простота:** легко реализовать и понять.  
✅ **Универсальность:** подходят для табличных данных, где нет пространственной/временной структуры.

❌ **Неэффективность для изображений/текстов:** не учитывают локальные зависимости (для этого есть CNN/RNN).  
❌ **Склонность к переобучению:** требуют регуляризации при большом числе параметров.

**Примеры задач**

* **Классификация:** Определение спама.
* **Регрессия:** Предсказание цены дома по характеристикам.

# Метод обратного распространения ошибки

Backpropagation — ключевой алгоритм обучения нейронных сетей, который **распространяет ошибку от выходного слоя к входному**, корректируя веса для минимизации функции потерь. Работает в два этапа:

1. **Прямой проход (Forward Pass)** — вычисление предсказания сети.
2. **Обратный проход (Backward Pass)** — расчёт градиентов и обновление весов.

**Шаг алгоритма:**

1. **Прямой проход**:
   * Вычислить выход сети для текущих весов.
   * Рассчитать ошибку (разницу между предсказанием и истинным значением).
2. **Обратный проход**:
   * Для **выходного слоя**:
     + Найти ∂L∂z(L)∂*z*(*L*)∂*L*​ (градиент по выходам).
   * Для **скрытых слоёв**:
     + Рекурсивно вычислить градиенты, используя цепное правило.
   * Обновить веса всех слоёв.

**Оптимизации:**

1. **Мини-батчи (Mini-batches)**:
   * Обновление весов на подмножествах данных (ускоряет обучение).
2. **Момент (Momentum)**:
   * Учёт предыдущих градиентов для подавления колебаний.
3. **Адаптивные методы (Adam, RMSprop)**:
   * Автоматическая настройка скорости обучения для каждого веса.

**Типичные проблемы:**

1. **Исчезающие градиенты**:
   * Проблема: Градиенты становятся слишком малыми в глубоких сетях.
   * Решение: использовать ReLU/LeakyReLU, остаточные связи (ResNet).
2. **Взрывающиеся градиенты**:
   * Проблема: Градиенты растут экспоненциально.
   * Решение: Нормализация градиентов (Gradient Clipping).
3. **Переобучение**:
   * Решение: Dropout, L2-регуляризация.

# Оптимизация при обучении нейронных сетей

**Нормализация данных**

Почему важно:

* Разные масштабы признаков → неравномерное обучение.

Методы:

* **StandardScaler**
* **MinMaxScaler**

**Управление градиентами**

Проблемы:

* **Исчезающие градиенты**: Слои ближе к входу не обучаются.
* **Взрывающиеся градиенты**: Нестабильность обучения.

Решение:

* **Batch Normalization**: Нормализует активации между слоями.
* **Gradient Clipping**: Ограничивает градиенты (например, max norm=1.0).

**Регуляризация**

Методы:

* **L1/L2-регуляризация**: Штраф за большие веса.
* **Dropout**: Случайное "выключение" нейронов во время обучения (обычно 20-50%).

**Оптимизация скорости обучения**

* **Learning Rate Schedule**:
  + Постепенное уменьшение (например, ReduceLROnPlateau).
* **Адаптивные оптимизаторы**:
  + **Adam**: Комбинация Momentum и RMSprop.
  + **Nadam**: Adam + Nesterov Momentum.

# Свёрточные нейронные сети

CNN — специализированный тип нейросетей для работы с **пространственными данными** (изображения, видео).

**Ключевые особенности:**

* Автоматическое выделение **локальных признаков** (грани, текстуры, объекты).
* Устойчивость к **сдвигам** и **масштабированию** изображений.

**Главные компоненты CNN**

1. **Свёрточный слой (Convolutional Layer)**
   * Применяет **фильтры (ядра)** к изображению.
   * Каждый фильтр обнаруживает определённый признак (например, границы).
   * **Параметры:**
     + kernel\_size (размер фильтра, например 3×3).
     + strides (шаг фильтра, обычно 1 или 2).
     + padding ('same' или 'valid').
2. **Пулинг слой (Pooling Layer)**
   * Уменьшает размерность, сохраняя важные признаки.
   * **Max Pooling**: Выбирает максимальное значение в области (чаще всего).
   * **Average Pooling**: Усреднение значений.
3. **Полносвязный слой (Fully Connected Layer)**
   * В конце сети — для классификации/регрессии.

**Архитектуры CNN**

* **LeNet-5** (первая CNN для распознавания цифр).
* **AlexNet** (победа в ImageNet 2012, ReLU, Dropout).
* **VGG** (глубокая сеть с 3×3 свёртками).
* **ResNet** (остаточные связи для обучения очень глубоких сетей).

**Плюсы и минусы CNN**

✅ **Автоматическое выделение признаков** (не нужен ручной feature engineering).  
✅ **Эффективность** для изображений, видео, аудио.

❌ **Требует много данных** для обучения с нуля.  
❌ **Вычислительно затратные** (особенно глубокие архитектуры).

# Нейросети для работы с последовательностями

**Основные понятия**

1. **Типы последовательностей**:
   * Временные ряды (финансы, сенсоры)
   * Текст (NLP)
   * Аудио (речь, музыка)
   * Последовательности действий (видео, игры)
2. **Ключевая проблема**:
   * Учет зависимостей между элементами последовательности
   * Обработка переменной длины входных данных

**Архитектуры нейросетей для последовательностей**

1. **Рекуррентные нейросети (RNN)**
   * Циклические связи для запоминания истории
   * Проблема: исчезающие градиенты
   * Варианты:
     + Простые RNN
     + LSTM (долгая краткосрочная память)
     + GRU (упрощенная версия LSTM)
2. **Сверточные сети для последовательностей (1D-CNN)**
   * Применяют 1D-свертки вдоль последовательности
   * Хорошо работают с локальными зависимостями
3. **Трансформеры (Transformers)**
   * Механизм внимания (attention)
   * Параллельная обработка всей последовательности
   * Современный стандарт для NLP

**Ключевые компоненты**

1. **Embedding-слой**:
   * Преобразование дискретных данных (слов, категорий) в векторы
   * Может обучаться или использовать предобученные (word2vec, GloVe)
2. **Механизмы памяти**:
   * Ячейки LSTM/GRU с gates (входными, выходными, забывания)
3. **Механизм внимания**:
   * Взвешенное суммирование элементов последовательности
   * Self-attention в трансформерах

**Плюсы и минусы**

✅ **RNN/LSTM**: Хороши для коротких последовательностей  
✅ **Трансформеры**: Лучшая производительность для длинных зависимостей  
❌ **RNN**: Медленные, проблемы с длинными последовательностями  
❌ **Трансформеры**: требуют много данных и ресурсов

# Языковые модели в машинном обучении

**Основные понятия**

1. **Что такое языковая модель?**
   * Вероятностная модель, предсказывающая следующее слово в последовательности на основе предыдущих.
   * Формально: вычисляет P(слово | контекст).
2. **Типы моделей**:
   * **Статистические** (N-граммы) — используют частоту встречаемости слов.
   * **Нейросетевые** (RNN, Transformer) — учитывают сложные зависимости.

**Ключевые архитектуры**

1. **Трансформеры**
   * **Self-attention**: взвешивает важность каждого слова в контексте.
   * **GPT** (Decoder-only): Генерация текста слева направо.
   * **BERT** (Encoder-only): Анализ двунаправленного контекста.
2. **Обучение**:
   * **Этап 1**: Предобучение на большом корпусе (Masked LM для BERT, next-word prediction для GPT).
   * **Этап 2**: Дообучение на конкретной задаче (Fine-tuning).

# Кластеризация в ML

1. **Что такое кластеризация?**
   * Метод **неконтролируемого обучения** (unsupervised) для группировки схожих объектов.
   * Цель: обнаружить **скрытые структуры** в данных без заранее известных меток.

A screenshot of a computer

AI-generated content may be incorrect.

**Как выбрать алгоритм?**

1. **Если известно число кластеров** → K-Means.
2. **Для данных с шумом/выбросами** → DBSCAN.
3. **Нужна интерпретация иерархии** → Иерархическая кластеризация.
4. **Сложная форма кластеров** → GMM или спектральная кластеризация.

**Этапы работы**

1. **Подготовка данных**:
   * Нормализация (например, StandardScaler).
   * Уменьшение размерности (PCA, t-SNE).
2. **Выбор метрики расстояния**:
   * Евклидово (для K-Means), косинусное (для текстов).
3. **Оценка качества**:
   * **Internal**: Индекс силуэта, Dunn index.
   * **External** (если есть истинные метки): ARI, NMI.

**Проблемы и решения**

* **Определение числа кластеров**:
  + Метод локтя (Elbow Method).
  + Анализ силуэта.
* **Масштабирование признаков**: Всегда нормализуйте данные!
* **Интерпретация результатов**: Визуализация через PCA или t-SNE.

# Метод главных компонент (PCA)

**PCA** — это метод снижения размерности данных, который преобразует признаки в новый набор ортогональных переменных (главных компонент), сохраняющих максимальную дисперсию исходных данных.

**Зачем нужен PCA?**

* Уменьшение числа признаков (борьба с проклятием размерности).
* Визуализация многомерных данных (например, 3D → 2D).
* Ускорение работы алгоритмов ML.
* Удаление коррелированных признаков.

**Как работает?**

1. **Стандартизация данных**:  
   Приводим все признаки к единой шкале (например, с помощью StandardScaler).
2. **Вычисление ковариационной матрицы**:  
   Показывает, как признаки коррелируют друг с другом.
3. **Собственные значения и векторы**:
   * Собственные векторы (components\_) — направления максимальной дисперсии.
   * Собственные значения (explained\_variance\_) — важность каждой компоненты.
4. **Проецирование данных**:  
   Умножаем исходные данные на собственные векторы.

**Ограничения PCA**

* **Линейность**: PCA работает только с линейными зависимостями. Для нелинейных данных используйте **Kernel PCA** или t-SNE.
* **Интерпретируемость**: Главные компоненты — это абстрактные направления, а не исходные признаки.
* **Чувствительность к выбросам**: Данные должны быть предварительно очищены.

# Предобработка данных

**1. Обработка пропущенных значений (Missing Data)**

* **Удаление строк/столбцов**: dropna() в Pandas.
* **Замена средним/медианой/модой**: fillna() для числовых и категориальных данных.
* **Предсказание пропусков**: Использование моделей (например, KNN или линейная регрессия).

**2. Кодирование категориальных признаков**

* **One-Hot Encoding**: pd.get\_dummies() или OneHotEncoder в sklearn.
* **Label Encoding**: LabelEncoder для порядковых категорий.
* **Target Encoding**: Замена категории средним значением целевой переменной.

**3. Масштабирование признаков**

* **Нормализация (Min-Max)**: Приводит данные к диапазону [0, 1] (MinMaxScaler).
* **Стандартизация (Z-score)**: Приводит данные к μ=0, σ=1 (StandardScaler).
* **Robust Scaling**: Устойчиво к выбросам (RobustScaler).

**4. Обработка выбросов (Outliers)**

* **Удаление**: По IQR или z-score.
* **Замена**: На граничные значения или медиану.
* **Трансформации**: Логарифмирование, корень для уменьшения влияния.

**5. Балансировка классов (для классификации)**

* **Undersampling**: Удаление примеров мажоритарного класса.
* **Oversampling**: Дублирование или генерация примеров (SMOTE).
* **Взвешивание классов**: class\_weight в моделях.

**6. Понижение размерности**

* **PCA**: Уменьшение числа признаков с сохранением дисперсии.
* **t-SNE / UMAP**: Визуализация многомерных данных.
* **Feature Selection**: Отбор признаков по важности (методы SelectKBest, RFE).

**7. Работа с временными рядами и текстом**

* **Для временных данных**: Создание лагов, скользящих средних.
* **Для текста**: TF-IDF, Word2Vec, BERT, токенизация.

**8. Другие методы**

* **Логарифмирование/степенные преобразования**: Для нормализации распределения.
* **Биннинг**: Группировка числовых значений в интервалы.

**Использование нормализации.**

**Что делает:**

* Приводит данные к диапазону **[0, 1]** (или другому, например, [-1, 1]).
* Чувствительна к выбросам (если есть аномальные значения, они "сожмут" основную часть данных).

**Когда использовать:**

* Когда данные имеют **ограниченный диапазон** (например, пиксели изображений 0-255).
* Когда алгоритмы требуют данных в определённом диапазоне (например, нейросети, KNN, градиентный спуск).

**Использование стандартизации.**

**Что делает:**

* Преобразует данные так, что среднее (**μ**) = **0**, стандартное отклонение (**σ**) = **1**.
* Менее чувствительна к выбросам, чем Min-Max.

**Когда использовать:**

* Когда данные **распределены нормально** (или близко к нормальному).
* Для алгоритмов, **чувствительных к масштабу** (логистическая регрессия, SVM, PCA).

**Ключевые отличия**

|  |  |  |
| --- | --- | --- |
| **Характеристика** | **Нормализация (Min-Max)** | **Стандартизация (Z-score)** |
| **Диапазон** | [0, 1] (или другой) | μ=0, σ=1 (может быть любой) |
| **Влияние выбросов** | Сильное | Слабое |
| **Распределение** | Не меняет форму | Делает ближе к нормальному |
| **Примеры алгоритмов** | KNN, нейросети, изображения | PCA, SVM, линейные модели |

**Выбор метода**

* Если данные **не имеют выбросов** и нужно строго задать диапазон → **нормализация**.
* Если данные **с выбросами** или нужна устойчивость → **стандартизация**.
* Для алгоритмов, **основанных на расстояниях** (KNN, K-means) часто лучше **нормализация**.
* Для **линейных моделей** и **методов, требующих нормальности** (PCA, LDA) → **стандартизация**.

# Прочее

**Разновидности признаков:**

* Численные
* Категориальные
* Бинарные

**Проблемы в данных:**

* Пропуски в данных
* Выбросы
* Ошибки разметки
* Data drift – смена данных с течением времени