# Лаб 14 (23)

Наивные байесовские классификаторы - это простые и быстрые алгоритмы для задач классификации (определения категории объекта, например, спам/не спам).

Они основаны на теореме Байеса (математическая формула для расчета вероятностей).

**Главная особенность и упрощение:** они «наивно» считают, что все признаки (характеристики) объекта независимы друг от друга внутри одного класса.

Пример: Если классифицируем email (спам/не спам), алгоритм считает, что наличие слова «скидка» не влияет на вероятность наличия слова «бесплатно», если мы уже знаем, что это спам. В реальности это часто не так, но это допущение сильно упрощает расчеты.

**Главное Преимущество - скорость**: Очень быстро обучаются и делают предсказания, так как обрабатывают каждый признак отдельно.

**Главный Недостаток:** Из-за «наивного» предположения о независимости признаков, точность может быть ниже, чем у более сложных моделей (вроде Logistic Regression), если признаки на самом деле сильно связаны.

**Основные Типы (в библиотеке scikit-learn)**

**GaussianNB**: Для непрерывных числовых данных (например, рост, температура). Считает среднее и разброс для каждого признака.

**BernoulliNB**: Для бинарных данных (0 или 1, есть признак или нет). Считает, как часто признак присутствует (не равен нулю) для каждого класса. Часто используется для текстов (присутствие/отсутствие слова).

**MultinomialNB**: Для счетных данных (например, сколько раз слово встретилось в тексте). Считает среднее количество или частоту признака для каждого класса. Часто работает лучше BernoulliNB для текстов, особенно длинных.

Параметр alpha (для BernoulliNB и MultinomialNB):

Это параметр сглаживания. Он добавляет небольшое искусственное «количество» ко всем признакам.

Зачем? Чтобы избежать проблем, если какой-то признак ни разу не встретился для какого-то класса в обучающих данных (иначе вероятность будет 0, что плохо).

**Эффект**: Чем больше alpha, тем «проще» модель, менее чувствительна к особенностям обучающих данных. Обычно не требует сильной настройки.

**Когда использовать?**

* Отличный стартовый вариант (baseline) для задачи классификации.
* Хорошо подходят для текстовых данных.
* Хорошо работают с данными высокой размерности (много признаков).
* Когда скорость обучения и предсказания очень важна.

В общем, это быстрые и простые классификаторы, основанные на вероятностях, с допущением о независимости признаков. Идеальны для текстов и как базовая модель, но могут уступать в точности более сложным алгоритмам.

# Лаб 15 (24)

**Деревья решений** - это модели машинного обучения для классификации (определение категории) и регрессии (предсказание числа).

Работают, задавая последовательность простых вопросов (тестов) по признакам данных.

**Структура похожа на блок-схему** или перевернутое дерево: корень -> узлы с вопросами -> ветви (ответы) -> листья (финальный ответ).

**Как строится дерево?**

Алгоритм рекурсивно делит данные на все более мелкие группы.

На каждом шаге выбирается лучший вопрос (тест) по одному из признаков, который максимально хорошо разделяет данные на группы, относящиеся к разным классам (для классификации) или имеющие разные значения (для регрессии). "Лучший" значит дающий наибольший прирост информации.

Тесты для непрерывных данных обычно имеют вид "Признак X <= Значение A?".

Процесс продолжается, пока в каждой конечной группе (листе) не останутся данные только одного класса (или очень похожие значения для регрессии). Такой лист называется чистым (pure).

**Как делается предсказание?**

Для нового объекта: начиная с корня, проходим по дереву, отвечая на вопросы в узлах.

Доходим до листа.

**Классификация**: Предсказанный класс - это большинство объектов, попавших в этот лист при обучении.

**Регрессия**: Предсказанное значение - это среднее значение целевой переменной у объектов, попавших в этот лист при обучении.

**Проблема**: Переобучение (Overfitting)

Если строить дерево до тех пор, пока все листья не станут "чистыми", оно идеально запомнит обучающие данные.

Такое дерево будет слишком сложным, уловит шум и случайные выбросы.

Оно будет плохо работать на новых, не виденных ранее данных (низкая обобщающая способность). Границы решения будут очень изрезанными.

**Борьба с переобучением: Обрезка (Pruning)**

Предварительная обрезка (pre-pruning): Остановить рост дерева раньше, не давая ему стать слишком сложным. Методы:

max\_depth: Ограничить максимальную глубину дерева (число вопросов подряд).

max\_leaf\_nodes: Ограничить максимальное количество листьев.

min\_samples\_leaf: Задать минимальное число объектов, которое должно быть в листе.

Именно pre-pruning в основном реализована в scikit-learn.

Пост-обрезка (post-pruning): Сначала построить полное (переобученное) дерево, а затем удалить или объединить "малоинформативные" узлы/ветви (в scikit-learn не реализовано напрямую).

**Важность признаков (Feature Importance)**

Деревья позволяют оценить, какие признаки были наиболее полезны для построения модели.

Важность признака - это число от 0 до 1, показывающее, насколько сильно этот признак уменьшал неопределенность (impurity) при разделениях по всему дереву. Сумма важностей всех признаков равна 1.

0 = признак не использовался, 1 = признак идеально предсказывает результат (редко).

Важно: Высокая важность не говорит, как именно признак влияет на результат (например, высокие значения признака ведут к классу А или Б). Она лишь показывает значимость признака для модели.

**Ограничение регрессии деревьями:**

Деревья не умеют экстраполировать, т.е. предсказывать значения вне диапазона целевой переменной, который был в обучающих данных.

Если в будущем появятся данные с характеристиками, ведущими к более высоким/низким значениям, чем были при обучении, дерево все равно предскажет максимальное/минимальное значение из обучающей выборки. Линейные модели, например, могут экстраполировать.

Преимущества деревьев:

Легко понять и визуализировать (особенно неглубокие деревья).

Не требуют масштабирования признаков (как MinMax Scaler или StandardScaler), так как каждый признак обрабатывается независимо.

Хорошо работают со смешанными типами данных (числовые, категориальные).

Недостатки деревьев:

Склонны к переобучению (требуется обрезка).

Даже с обрезкой могут иметь невысокую обобщающую способность по сравнению с другими методами.

Нестабильны: Небольшие изменения в данных могут привести к построению совершенно другого дерева.

Не могут экстраполировать в задачах регрессии.

1. **Дерево классификации животных** (Рис. 1) Что видим: Простую древовидную структуру. Начинается с корневого узла ("Есть перья?"). От него идут две ветви ("Истина", "Ложь"), ведущие к следующим узлам с вопросами ("Летать?", "Есть плавники?") или к терминальным узлам-листьям ("Ястреб", "Пингвин", "Дельфин", "Медведь"), которые содержат ответ (предсказанный класс животного).

**Объяснение**: Это базовая иллюстрация концепции дерева решений. Показывает, как последовательность простых бинарных ("да/нет") вопросов может привести к классификации объекта. Каждый узел (кроме листьев) - это тест (вопрос по признаку), а листья - это итоговые классы.

1. **Пошаговое построение дерева** (Рис. 2, 3, 4, 5 - левая часть) Что видим: Серию графиков (генерируемых mglearn.plots.plot\_tree\_progressive). На первом (Рис. 2) показан исходный набор данных "two\_moons" (два класса, синие кружки и красные треугольники). На последующих графиках (Рис. 3, 4, 5 слева) показано, как пространство признаков рекурсивно разделяется на прямоугольные области с помощью горизонтальных и вертикальных линий. С каждым шагом (увеличением глубины дерева) добавляются новые линии, делая разделение все более детальным. Области окрашены в цвет преобладающего класса.

**Объяснение**: Это демонстрация процесса построения дерева. Алгоритм на каждом шаге ищет "лучшее" (наиболее информативное) разделение (тест вида "признак <= порог"), которое максимально хорошо отделяет один класс от другого в текущей области. Эти разделения всегда параллельны осям координат. Видно, как с ростом глубины дерево все точнее подстраивается под обучающие данные, что может привести к переобучению (как на Рис. 5 слева, где границы очень изрезаны).

1. Структура дерева решений (Рис. 3, 4, 5 - правая часть) Что видим: Соответствующие древовидные структуры для разделений, показанных слева. Каждый узел показывает:

* Тест (например, X[1] <= 0.0596).
* samples: Количество точек данных, попавших в этот узел.
* value: Распределение этих точек по классам (например, [50, 50] - 50 точек класса 0, 50 точек класса 1).
* class (в более поздних визуализациях): Класс, который будет предсказан для этого узла (если бы он был листом).

**Объяснение**: Это графическое представление самой модели дерева решений, соответствующее разделениям на графиках слева. Показывает логику "если-то", которую использует модель. Видно, как данные "протекают" по дереву в зависимости от результатов тестов. На Рис. 5 справа показан лишь фрагмент очень глубокого и сложного дерева.

1. **Визуализация дерева для рака груди** (Рис. 6) Что видим: Визуализацию обрезанного дерева решений (max\_depth=4), обученного на реальном датасете Breast Cancer. Используются реальные имена признаков (worst radius, texture error и т.д.) и классов (malignant, benign). Узлы окрашены для указания преобладающего класса, интенсивность цвета может отражать "чистоту" узла (насколько один класс доминирует).

**Объяснение**: Показывает, как выглядит дерево для реальной задачи. Позволяет интерпретировать модель: можно проследить путь для конкретного примера и понять, почему был сделан тот или иной прогноз. Видно, какие признаки и пороговые значения модель сочла важными на верхних уровнях. Даже при глубине 4 дерево уже довольно ветвистое.

1. **Важность признаков** (Рис. 7) Что видим: Горизонтальную столбчатую диаграмму. Каждому признаку из датасета Breast Cancer соответствует столбец. Длина столбца показывает важность (feature importance) этого признака, вычисленную деревом решений (в данном случае, необрезанным).

**Объяснение**: Это способ суммировать информацию о полезности признаков для модели, не просматривая все дерево. Важность показывает, насколько сильно признак способствовал уменьшению неопределенности (примеси) при построения дерева. Здесь видно, что признак worst radius имеет наибольшую важность, что согласуется с тем, что он часто используется для разделения в верхних узлах дерева (как на Рис. 6). Важно помнить, что это не говорит о направлении влияния признака, только о его значимости для данной модели дерева.

1. **Немонотонная взаимосвязь - Границы** (Рис. 8) Что видим: Двумерный набор данных, где класс объекта зависит от признака X[1] нелинейно (немонотонно): точки класса 1 (зеленые треугольники) находятся в "середине" по оси Y, а точки класса 0 (синие кружки) - по краям. Показаны границы решения, найденные деревом - они снова состоят из горизонтальных и вертикальных линий.

**Объяснение**: Демонстрирует способность деревьев улавливать сложные, нелинейные зависимости. Линейная модель не смогла бы хорошо разделить эти классы одной прямой. Дерево же, используя несколько разделений по оси Y, успешно изолирует области разных классов.

1. **Немонотонная взаимосвязь - Структура дерева** (Рис. 9) Что видим: Дерево решений, которое создает границы, показанные на Рис. 8.

**Объяснение**: Показывает, как дерево достигает немонотонного разделения. Видно, что оно использует один и тот же признак (X[1]) несколько раз на разных уровнях с разными порогами (<= -5.8141, <= 5.3475). Это позволяет ему "вырезать" среднюю область. Важность признака X[1] для этого дерева будет равна 1.0, а X[0] - 0.0, так как X[0] вообще не использовался.

1. **Исторические цены на RAM (Рис. 10)** Что видим: График изменения цен на оперативную память (RAM) с течением времени (с 1950-х по ~2015). Ось Y (цена) имеет логарифмическую шкалу.

**Объяснение**: Это исходные данные для задачи регрессии. Логарифмическая шкала используется потому, что цены падали экспоненциально. На логарифмической шкале эта экспоненциальная зависимость выглядит примерно как прямая линия (с некоторыми колебаниями), что удобно для моделирования.

1. **Сравнение прогнозов цен на RAM (Рис. 11)** Что видим: Сравнение реальных цен (обучающие и тестовые данные) с прогнозами двух моделей: дерева решений (DecisionTreeRegressor) и линейной регрессии (LinearRegression). Обе модели обучались на данных до 2000 года (на логарифме цены). Ось Y снова логарифмическая.

**Объяснение**: Ключевой график, иллюстрирующий недостаток деревьев регрессии - неумение экстраполировать.

Линейная регрессия уловила общий нисходящий тренд и продолжила его в будущее (экстраполировала), давая разумные (хотя и не идеальные) прогнозы на тестовом периоде (после 2000 года).

Дерево решений идеально подогналось под обучающие данные (до 2000 года). Но для всех дат после 2000 года оно просто предсказывает последнее известное ему значение из обучающей выборки. Оно не может генерировать значения ниже тех, что видело при обучении. Это типичное поведение для деревьев регрессии при прогнозировании за пределы диапазона обучающих данных.

# Лаб 16 (25)

**Ансамбли (Ensembles)** - это методы машинного обучения, которые объединяют несколько моделей (часто одного типа, например, деревьев решений) для получения более точного и стабильного результата, чем у любой из одиночных моделей.

Два популярных и эффективных ансамбля на основе деревьев:**Случайный лес** и **Градиентный бустинг.**

**Случайный лес (Random Forest)**

**Идея**: Обучить много разных деревьев решений и усреднить их предсказания. Усреднение помогает снизить переобучение, свойственное одиночным деревьям.

**Как достигается "разность" деревьев? (Рандомизация):**

* **Бутстреп-выборки (Bootstrap Samples):**Каждое дерево обучается на случайной подвыборке исходных данных, взятой с возвращением. Размер подвыборки равен размеру исходных данных, но из-за возвращения некоторые объекты попадут несколько раз, а некоторые (около 1/3) не попадут вовсе.
* **Случайные подмножества признаков (Random Subsets of Features):** При поиске лучшего разделения в каждом узле дерева рассматривается не все признаки, а только их случайное подмножество (размер контролируется параметром max\_features).

**Предсказание**:

* **Классификация**: Каждое дерево "голосует" за свой класс (или предсказывает вероятности). Итоговый класс - тот, за который проголосовало большинство деревьев (или класс с максимальной средней вероятностью - "мягкое голосование").
* **Регрессия**: Предсказания всех деревьев усредняются.

**Параметры**:

* **n\_estimators:** Количество деревьев в лесу. Больше - лучше (до определенного предела), но дольше обучается и требует больше памяти.
* **max\_features**: Размер случайного подмножества признаков для каждого узла. Важный параметр для контроля разнообразия деревьев. Значения по умолчанию (sqrt(n\_features) для классификации, n\_features для регрессии) часто работают хорошо.
* **Параметры обрезки деревьев** (max\_depth, min\_samples\_leaf и т.д.): Можно использовать для дополнительного контроля сложности отдельных деревьев.

**Преимущества**: Высокая точность, устойчивость к переобучению (лучше одиночного дерева), не требует масштабирования данных, хорошо работает "из коробки", легко распараллеливается.

**Недостатки**: Менее интерпретируем, чем одно дерево; требует больше памяти и времени на обучение/предсказание, чем линейные модели; может плохо работать на очень разреженных данных (как тексты).

**Градиентный бустинг деревьев решений (Gradient Boosting Decision Trees - GBDT)**

**Идея**: Строить деревья последовательно, где каждое следующее дерево пытается исправить ошибки (остатки) предыдущих. Ансамбль строится итеративно.

**Как работает:**

* Строится первое (обычно очень простое) дерево.
* Вычисляются ошибки (разница между реальным значением и предсказанием) этой модели.
* Строится второе дерево, которое обучается предсказывать эти ошибки.
* Предсказания первого и второго дерева (с некоторым весом) объединяются.
* Вычисляются новые ошибки.
* Строится третье дерево, предсказывающее новые ошибки, и т.д.

**Ключевые особенности:**

* Деревья обычно очень неглубокие (max\_depth часто 1-5). Это "слабые ученики" (weak learners).
* Нет случайности (по умолчанию) как в Случайном лесе. Построение детерминировано.
* Последовательное построение: Нельзя распараллелить так же легко, как Случайный лес.

**Параметры**:

* n\_estimators: Количество деревьев (итераций). Слишком большое значение может привести к переобучению, в отличие от Случайного леса.
* learning\_rate (скорость обучения): Контролирует "вес" каждого нового дерева в ансамбле. Меньшее значение требует больше деревьев (n\_estimators) для достижения той же сложности, но часто дает лучшие результаты (меньше переобучение). Важный параметр для настройки!
* max\_depth (или max\_leaf\_nodes): Контролирует сложность каждого отдельного дерева. Обычно используют маленькие значения.

**Преимущества**: Часто дает самую высокую точность из многих алгоритмов, очень мощный метод.

**Недостатки**: Чувствителен к настройке параметров (n\_estimators, learning\_rate, max\_depth тесно связаны); обучение может быть долгим (т.к. последовательное); более склонен к переобучению, чем Случайный лес, если параметры не настроены; как и RF, плохо работает на разреженных данных.

**Объяснение графиков (Рисунки из Лабораторной работы №25):**

1. **Границы решений 5 деревьев и случайного леса (Рис. 1).**

**Что видим**: 6 графиков на данных make\_moons. Первые 5 показывают границы решения, найденные каждым из пяти отдельных деревьев, входящих в состав случайного леса. Шестой график показывает итоговую границу решения всего ансамбля (случайного леса), полученную усреднением предсказаний первых пяти.

**Объяснение**: Иллюстрирует ключевую идею Случайного леса. Видно, что границы отдельных деревьев довольно разные (из-за рандомизации при их построении) и могут быть не очень гладкими или даже ошибочными в некоторых областях. Однако итоговая граница ансамбля получается гораздо более гладкой и лучше соответствует истинной структуре данных, чем у любого отдельного дерева. Усреднение сглаживает ошибки и выбросы отдельных моделей.

1. **Важности признаков (Случайный лес, Breast Cancer) (Рис. 2)**

**Что видим:** Горизонтальная столбчатая диаграмма важности признаков для датасета Breast Cancer, вычисленная моделью Случайного леса из 100 деревьев.

**Объяснение**: Показывает, какие признаки Случайный лес считает наиболее важными в среднем по всем деревьям ансамбля. В отличие от одиночного дерева (Рис. 7 из ЛР24), Случайный лес присваивает ненулевую важность гораздо большему числу признаков. Это происходит потому, что из-за случайного выбора признаков в узлах разные деревья "узнают" о полезности разных признаков. Важности, полученные от Случайного леса, обычно считаются более надежными и стабильными, чем от одного дерева. Здесь worst perimeter и worst concave points выделяются как наиболее важные.

1. **Важности признаков (Градиентный бустинг, Breast Cancer) (Рис. 3)**

**Что видим:** Горизонтальная столбчатая диаграмма важности признаков для датасета Breast Cancer, вычисленная моделью Градиентного бустинга (с max\_depth=1).

**Объяснение:** Показывает важность признаков с точки зрения модели градиентного бустинга. Результаты могут отличаться от Случайного леса. Здесь видно, что градиентный бустинг (в данной конфигурации) также выделяет worst perimeter, но, например, worst texture и mean concave points получают большую важность, чем в RF. Интересно, что градиентный бустинг может полностью игнорировать некоторые признаки (присваивать им нулевую важность), если они не помогают исправить ошибки на ранних этапах построения ансамбля.

# Лаб 17 (26)

**Метод Опорных Векторов (SVM):**Это мощный метод обучения с учителем для классификации (SVC) и регрессии (SVR). Документ фокусируется на SVC.

**Ядерный SVM (Kernel SVM):** Это расширение линейного SVM, позволяющее строить сложные, нелинейные границы решений. Вместо построения простой гиперплоскости в исходном пространстве, SVM использует "ядерный трюк".

**Идея нелинейности:** Если классы нельзя разделить прямой линией (или плоскостью) в исходном пространстве признаков, можно попробовать:

**Добавить новые признаки:** Например, полиномиальные (x², y², xy) или другие комбинации исходных. В новом, расширенном пространстве, данные могут стать линейно разделимыми (как показано в примере с добавлением признак1\*\*2).

**Проблема:**Добавление многих признаков (особенно полиномов высоких степеней или взаимодействий) может быть вычислительно очень дорогим и привести к огромному количеству измерений.

**Ядерный трюк (Kernel Trick):**

Это математический прием, который позволяет SVM работать так, как будто данные были перенесены в пространство очень высокой (иногда бесконечной) размерности с новыми нелинейными признаками, но без явного вычисления координат точек в этом новом пространстве.

Вместо этого вычисляются скалярные произведения (или расстояния) между точками в этом неявном пространстве с помощью функции ядра (kernel function).

**Популярные ядра:**

* Полиномиальное ядро: Вычисляет все полиномиальные комбинации до определенной степени.
* Ядро RBF (Радиальная Базисная Функция) или Гауссовское ядро: Самое популярное. Соответствует бесконечномерному пространству признаков. Измеряет "схожесть" точек на основе евклидова расстояния; точки, близкие в исходном пространстве, считаются схожими.

**Принцип работы SVM с ядром RBF:**

* Опорные векторы (Support Vectors): Это точки обучающей выборки, которые лежат на границе классов или были ошибочно классифицированы при определенном уровне "мягкости" границы (параметр C). Именно эти точки определяют положение и форму границы решения. Остальные точки не влияют.
* Предсказание для новой точки: Вычисляется "расстояние" (через RBF ядро) от новой точки до каждого опорного вектора. Решение о классе принимается на основе взвешенной суммы этих расстояний, где веса (dual\_coef\_) показывают важность каждого опорного вектора.
* Формула Гауссовского ядра: k\_rbf(x1, x2) = exp(-gamma \* ||x1 - x2||^2), где ||x1 - x2||^2 - квадрат евклидова расстояния, а gamma - параметр ядра.

**Параметры RBF SVM:**

* gamma: Определяет "ширину" влияния одного опорного вектора.
* Маленькая gamma: Широкое влияние, гладкая, простая граница (низкая сложность модели).
* Большая gamma: Узкое, локальное влияние, граница сильно подстраивается под отдельные точки (высокая сложность модели, риск переобучения).
* C: Параметр регуляризации (как в линейных моделях). Контролирует "штраф" за неправильную классификацию точек.
* Маленький C: Большой штраф, модель старается быть проще, граница более "жесткая", допускает ошибки на опорных векторах.
* Большой C: Маленький штраф, модель сильно подстраивается под опорные векторы, пытается классифицировать их правильно, граница может стать более сложной и изогнутой (риск переобучения).
* Взаимосвязь: C и gamma сильно влияют друг на друга. Высокие значения обоих приводят к очень сложным моделям. Их нужно настраивать совместно.

**Важность Масштабирования:**

Ядерный SVM (особенно с RBF ядром) чрезвычайно чувствителен к масштабу признаков. RBF ядро использует евклидово расстояние. Если один признак имеет диапазон [0, 1], а другой [0, 1000], то второй признак будет доминировать при расчете расстояния, и влияние первого будет почти незаметно.

Необходимо приводить все признаки к одному масштабу перед обучением SVM (например, с помощью MinMaxScaler [0, 1] или StandardScaler [среднее 0, дисперсия 1]). Это критически важно для получения хороших результатов.

**Преимущества SVM:**

* Позволяют строить очень сложные, нелинейные границы.
* Эффективны в пространствах высокой размерности.
* Хорошо работают, когда число признаков сравнимо или больше числа объектов.
* Мощный метод с хорошей точностью на разнообразных данных (при правильной настройке и масштабировании).

**Недостатки SVM:**

* Плохо масштабируются с ростом числа объектов (время обучения может быть O(N²) или O(N³)). Проблематично для >100,000 объектов.
* Требуют тщательной предобработки данных (масштабирования).
* Требуют тщательной настройки параметров (C, gamma, тип ядра).
* Модель трудно интерпретировать ("черный ящик") - сложно понять, почему было принято конкретное решение.

**Объяснение графиков (Рисунки из Лабораторной работы №26):**

* **Рис. 1:**Исходные данные make\_blobs, где 4 кластера объединены в 2 класса. Видно, что классы (синие кружки и зеленые треугольники) нельзя разделить одной прямой линией.
* **Рис. 2:**Результат применения LinearSVC к данным с Рис. 1. Прямая линия плохо разделяет классы, много ошибок.
* **Рис. 3:**Те же данные, но теперь в 3D. К исходным двум признакам добавлен третий - квадрат второго признака (признак1 \*\* 2). Видно, что в этом новом пространстве классы "поднялись" на разную высоту и теперь их можно разделить плоскостью.
* **Рис. 4:**Та же 3D визуализация, но теперь добавлена разделяющая плоскость, найденная LinearSVC в этом 3D пространстве. Плоскость успешно отделяет синие точки от красных треугольников.
* **Рис. 5:** Граница решения, соответствующая плоскости с Рис. 4, но спроецированная обратно на исходное 2D пространство признаков (Признак 0, Признак 1). Видно, что граница стала нелинейной (похожа на параболу/эллипс). Это демонстрирует, как добавление нелинейных признаков позволяет линейной модели создавать нелинейные границы в исходном пространстве.
* **Рис. 6:**Пример работы SVM с RBF ядром на другом, специально созданном 2D датасете (make\_handcrafted\_dataset). Показывает гладкую, нелинейную границу решения, найденную RBF SVM. Точки с толстой обводкой - это опорные векторы, которые определяют границу.
* **Рис. 7:** Сетка 3x3 графиков, иллюстрирующая влияние параметров C и gamma на границу решения RBF SVM и выбор опорных векторов.

По горизонтали (слева направо): gamma увеличивается (0.1, 1, 10). Граница становится всё более сложной, фокусируется на отдельных точках.

По вертикали (сверху вниз): C увеличивается (0.1, 1, 1000). Модель сильнее подстраивается под опорные векторы, граница становится более изогнутой, чтобы правильно их классифицировать.

Видно, как комбинация параметров влияет на сложность модели.

* **Рис. 8:** График минимальных и максимальных значений для каждого признака в датасете Breast Cancer (обучающая выборка). Ось Y - логарифмическая. Видно, что разброс значений между признаками огромен (на несколько порядков). Это наглядно показывает, почему масштабирование данных критически важно для SVM.

# Лаб 18 (27)

**Нейронные сети** - это семейство алгоритмов машинного обучения, вдохновленное (очень упрощенно) работой человеческого мозга.

**Многослойный персептрон (Multilayer Perceptron, MLP)** - это классический тип нейронной сети прямого распространения (feedforward), рассматриваемый в этой работе. Его можно считать обобщением линейных моделей.

**Термин "Глубокое обучение"** (Deep Learning) относится к нейронным сетям с большим количеством слоев.

**Структура MLP:**

* **Входной слой (Input Layer):**Принимает исходные признаки данных (как в линейной модели). Каждый узел соответствует одному признаку.
* **Скрытые слои (Hidden Layers):** Один или несколько промежуточных слоев узлов (нейронов). Это ключевое отличие от линейных моделей. Каждый узел в скрытом слое вычисляет взвешенную сумму сигналов от всех узлов предыдущего слоя, а затем применяет к этой сумме нелинейную функцию активации.
* **Выходной слой (Output Layer):** Последний слой, который выдает итоговый результат (предсказание класса или числовое значение). Он также получает взвешенные суммы от последнего скрытого слоя и применяет функцию активации (для регрессии может не применяться).
* **Веса (Weights):** Коэффициенты на соединениях между узлами соседних слоев. Именно эти веса модель обучает на данных.

**Почему нужны нелинейные функции активации?**

Если бы между слоями были только взвешенные суммы (линейные операции), то вся сеть, независимо от числа слоев, была бы эквивалентна одной линейной модели.

Нелинейные функции активации (применяемые к выходу каждого скрытого нейрона) позволяют сети изучать сложные, нелинейные зависимости в данных.

**Популярные функции активации:**

* ReLU (Rectified Linear Unit): f(x) = max(0, x). Простая и эффективная, отсекает отрицательные значения. Используется по умолчанию в MLPClassifier.
* Tanh (Гиперболический тангенс): f(x) = tanh(x). Выдает значения в диапазоне [-1, 1]. Более гладкая, чем ReLU.
* Сигмоида (Sigmoid): Раньше была популярна, но сейчас реже используется в скрытых слоях из-за проблем с затуханием градиентов. Выдает значения [0, 1].

**Настройка сложности MLP:** Сложность (и способность к переобучению) можно контролировать через:

* Количество скрытых слоев: Больше слоев -> потенциально более сложные зависимости, но сложнее обучать ("глубокое обучение").
* Количество нейронов (узлов) в каждом скрытом слое: Больше нейронов -> больше весов, выше гибкость модели. Типичные значения от 10 до тысяч.
* Регуляризация (параметр alpha): L2-штраф (как в Ridge регрессии), который "сжимает" веса к нулю, делая модель проще и устойчивее к переобучению. Большее alpha = сильнее регуляризация.
* Функция активации: Разные функции дают разную "форму" нелинейности.

**Обучение нейронной сети:**

* Процесс поиска оптимальных весов сложен и итеративен. Используются алгоритмы оптимизации, основанные на градиентном спуске (например, adam, lbfgs, sgd).
* Случайная инициализация: Веса перед обучением инициализируются случайными значениями. Это важно, но также означает, что результат обучения может немного отличаться при разных запусках (если не зафиксировать random\_state).
* Масштабирование данных: Как и SVM, нейронные сети очень чувствительны к масштабу признаков. Данные необходимо масштабировать (например, StandardScaler - среднее 0, ст. отклонение 1), чтобы все признаки имели схожий диапазон и обучение проходило корректно.

**Параметр solver:**Задает алгоритм оптимизации для поиска весов.

* adam: Эффективен на больших данных, работает хорошо по умолчанию, но требует масштабирования данных.
* lbfgs: Хорош для малых датасетов, может сходиться быстрее, менее чувствителен к масштабу, но может требовать больше памяти.
* sgd: Стохастический градиентный спуск. Основа многих продвинутых техник, но требует тщательной настройки доп. параметров (скорость обучения и др.).

**Преимущества MLP:**

* Способность изучать очень сложные нелинейные зависимости.
* При достаточных данных, времени и настройке могут достигать очень высокой точности, часто превосходя другие модели.
* Основа для глубокого обучения и работы с неструктурированными данными (изображения, звук, текст).

**Недостатки MLP:**

* Требуют тщательной предобработки данных (масштабирования).
* Чувствительны к настройке многих параметров (архитектура, alpha, solver, learning\_rate для sgd и т.д.). Настройка может быть "искусством".
* Обучение может быть очень долгим, особенно для больших сетей и данных.
* Склонны к переобучению.
* Трудно интерпретировать ("черный ящик"). Сложно понять, почему сеть приняла то или иное решение, глядя на веса.
* Результат зависит от случайной инициализации весов.

**Объяснение графиков (Рисунки из Лабораторной работы №27):**

* **Рис. 1:** Визуализация логистической регрессии как простейшей "сети". Входные узлы (признаки x[0]...x[p]) напрямую соединены с выходным узлом (ŷ) через веса (w[0]...w[p]). Показывает базовую идею взвешенной суммы.
* **Рис. 2:**Схема MLP с одним скрытым слоем. Входы (x) соединены с узлами скрытого слоя (h), а те, в свою очередь, с выходом (ŷ). Показывает промежуточный этап вычислений.
* **Рис. 3:** Графики двух нелинейных функций активации: tanh (сигмоидальная кривая от -1 до 1) и ReLU (ломаная линия: 0 для x<0, x для x>=0). Эти функции вносят нелинейность в модель.
* **Рис. 4:** Схема MLP с двумя скрытыми слоями. Показывает возможность добавления глубины сети.
* **Рис. 5:** Граница решения MLP (100 скрытых нейронов, ReLU) на данных make\_moons. Граница нелинейная, довольно гладкая, хорошо разделяет классы.
* **Рис. 6:**Граница решения MLP (10 скрытых нейронов, ReLU). С меньшим числом нейронов граница стала менее гладкой, более "ломаной", состоящей из меньшего числа линейных сегментов (по числу нейронов).
* **Рис. 7** (\*поправка: в коде это ReLU, не tanh): Граница решения MLP (2 скрытых слоя по 10 нейронов, ReLU). Добавление второго слоя позволяет создать еще более сложную границу, чем на Рис. 6, даже с тем же общим числом нейронов.
* **Рис. 8:**Граница решения MLP (2 скрытых слоя по 10 нейронов, tanh). С функцией активации tanh граница выглядит более гладкой, чем с ReLU (Рис. 7\*), но сохраняет сложную форму.
* **Рис. 9:** Сетка графиков, показывающая влияние регуляризации alpha и числа скрытых нейронов на границу решения.

Верхний ряд: 2 слоя по 10 нейронов. Нижний ряд: 2 слоя по 100 нейронов.

Столбцы (слева направо): alpha увеличивается (0.0001, 0.01, 0.1, 1).

Видно, что при маленьком alpha (слабая регуляризация) граница более сложная, подстраивается под данные. При большом alpha граница становится значительно проще и глаже, так как веса "сжимаются" к нулю. Большее число нейронов (нижний ряд) позволяет строить более сложные границы при низком alpha.

* **Рис. 10:**Сетка графиков, показывающая влияние случайной инициализации весов (random\_state). Все параметры сети одинаковы, меняется только стартовое значение генератора случайных чисел. Видно, что итоговая граница решения может заметно отличаться в зависимости от начальных весов, особенно для сложных моделей или небольших датасетов.
* **Рис. 11:**Теплокарта весов между входным слоем (оси Y, 30 признаков Breast Cancer) и первым скрытым слоем (оси X, 100 нейронов) для обученной модели MLP (на масштабированных данных, с alpha=1). Цвет ячейки показывает величину веса: светлый - большой положительный, темный - большой отрицательный, зеленый - близкий к нулю.

Объяснение: Попытка заглянуть внутрь модели. Позволяет увидеть, какие входные признаки имеют сильные связи (положительные или отрицательные) с какими нейронами скрытого слоя. Однако интерпретировать эту матрицу целиком очень сложно. Можно заметить, что некоторые признаки (строки) имеют в целом более яркие/темные цвета, что может указывать на их большую важность, а некоторые нейроны (столбцы) сильнее реагируют на определенные группы признаков.

# Лаб 19 (28)

**Зачем нужна неопределенность?** Часто недостаточно знать только предсказанный класс, важно понимать, насколько модель уверена в своем прогнозе.

Это критично в областях вроде медицины (ошибка ложноотрицательного диагноза может быть фатальна) или финансов.

Понимание неопределенности помогает оценить риски и принять более информированные решения.

**Методы оценки неопределенности в scikit-learn:**

* **decision\_function**: Возвращает числовое значение для каждого образца. Чем дальше значение от 0, тем увереннее модель. Знак определяет предсказанный класс в бинарной классификации.
* **predict\_proba:** Возвращает вероятность принадлежности образца к каждому классу. Значения всегда между 0 и 1, сумма вероятностей по всем классам для одного образца равна 1.

Не все классификаторы поддерживают оба метода, но большинство - хотя бы один. GradientBoostingClassifier поддерживает оба.

**decision\_function** (Решающая функция): Бинарная классификация: Возвращает массив формы (n\_samples,).

* **Значение > 0:** Модель предсказывает "положительный" класс (второй элемент в атрибуте model.classes\_).
* **Значение < 0:** Модель предсказывает "отрицательный" класс (первый элемент в model.classes\_).
* **Значение = 0:** Точка лежит точно на границе решения.
* **Абсолютное значение** показывает расстояние до границы решения (чем больше, тем увереннее).
* **Масштаб значений произволен**и зависит от модели и данных, что затрудняет прямую интерпретацию "уверенности" в виде вероятности.

Мультиклассовая классификация: Возвращает массив формы (n\_samples, n\_classes).

* Каждый столбец соответствует одному классу.
* Значение в ячейке [i, j] - это "мера уверенности" модели в том, что образец i принадлежит классу j.
* Предсказанный класс - это тот, который имеет максимальное значение в строке (находится с помощью np.argmax(..., axis=1)).

**predict\_proba** (Прогнозирование вероятностей):

* Возвращает массив формы (n\_samples, n\_classes) как для бинарной, так и для мультиклассовой классификации.
* Значение в ячейке [i, j] - это оценка вероятности того, что образец i принадлежит классу j.
* Значения всегда в диапазоне [0, 1].
* Сумма значений в каждой строке всегда равна 1.
* Интерпретировать легче, чем decision\_function, так как это вероятности. Класс с вероятностью > 0.5 (в бинарном случае) или с максимальной вероятностью (в мультиклассовом) будет предсказан.
* Калибровка (Calibration): Точность предсказанных вероятностей (насколько они соответствуют реальной частоте событий) зависит от модели и данных. Хорошо откалиброванная модель, дающая прогноз с уверенностью 70%, будет права примерно в 70% случаев. Переобученные модели часто дают слишком уверенные (близкие к 0 или 1), но неточные вероятности.

**Связь с predict и classes\_:**

Метод predict по сути реализует логику выбора класса на основе decision\_function (по знаку или argmax) или predict\_proba (по argmax).

Важно помнить, что argmax возвращает индексы (0, 1, 2,...). Чтобы получить фактические имена классов (особенно если они строки или не идут подряд с нуля), нужно использовать эти индексы для извлечения значений из атрибута model.classes\_. Атрибут model.classes\_ хранит уникальные метки классов в том порядке, в котором их "видела" модель при обучении.

**Объяснение графиков (Рисунки из Лабораторной работы №28):**

**Рис. 1:**Сравнение визуализаций для GBDT на данных make\_circles.

**Левый график:** Стандартная граница принятия решений. Показывает, какая область пространства к какому классу будет отнесена (красная/синяя).

**Правый график:**Теплокарта значений decision\_function. Цвет показывает значение решающей функции: темно-красный - высокое положительное значение (очень уверенно класс 'red'), темно-синий - высокое отрицательное значение (очень уверенно класс 'blue'), светлые/зеленоватые цвета - значения около нуля (область неопределенности, близко к границе). Позволяет увидеть не только границу, но и "ландшафт уверенности" модели.

**Рис. 2:** Снова сравнение визуализаций для GBDT на make\_circles.

Левый график: Граница принятия решений (та же, что на Рис. 1 слева).

Правый график: Теплокарта вероятностей predict\_proba для класса 'red'. Цвет показывает вероятность принадлежности точки к классу 'red': ярко-красный - вероятность близка к 1, ярко-синий - вероятность близка к 0 (т.е. высокая вероятность класса 'blue'), переходные цвета - вероятности около 0.5. Эта визуализация часто более интуитивно понятна, чем decision\_function, так как оперирует вероятностями [0, 1]. Границы здесь выглядят четче.

**Рис. 3:** Сравнение различных классификаторов scikit-learn на нескольких синтетических наборах данных (взято с сайта scikit-learn). Показывает, как разные алгоритмы (Nearest Neighbors, Linear SVM, RBF SVM, Decision Tree, RandomForest, Neural Net, AdaBoost, Naive Bayes, QDA) строят границы решений и как выглядит их "уверенность" (через decision\_function или predict\_proba) на разных типах данных. Полезно для понимания сильных и слабых сторон разных моделей.

# Лаб 20 (29)

**Обучение без учителя** - это тип машинного обучения, где используются только входные данные (признаки), без каких-либо заранее известных "правильных ответов" (меток или целевых переменных).

**Цель** - найти структуру, закономерности или новое представление в самих данных.

Нет "учителя", который бы говорил алгоритму, правильный ли ответ он нашел.

**Основные типы Обучения без учителя:**

**Неконтролируемые преобразования** (Unsupervised Transformations):

* Создают новое представление данных. Цель - сделать данные более удобными для обработки человеком или другими алгоритмами МО.
* Сокращение размерности (Dimensionality Reduction): Уменьшение количества признаков при сохранении основной информации (например, для визуализации данных высокой размерности на 2D/3D графике или для сжатия данных).
* Поиск компонент/Выделение признаков: Нахождение скрытых составляющих или тем в данных (например, темы в текстовых документах, компоненты изображений лиц).

**Кластеризация (Clustering):**

* Разбиение данных на группы (кластеры) схожих между собой объектов. Объекты внутри одного кластера должны быть похожи, а объекты из разных кластеров - отличаться.
* Пример: Группировка фотографий людей по лицам, сегментация клиентов по покупательскому поведению.

**Проблемы Обучения без учителя:**

* Сложность оценки результата: Так как нет "правильных ответов", трудно автоматически оценить, насколько хорошо алгоритм справился с задачей (например, насколько "правильно" данные разбиты на кластеры). Часто требуется ручная проверка и интерпретация экспертом.
* Алгоритмы часто используются для разведочного анализа данных (Exploratory Data Analysis - EDA), чтобы лучше понять данные перед применением обучения с учителем.
* Могут служить этапом предобработки для обучения с учителем (например, сокращение размерности может улучшить производительность или точность последующего классификатора).

**Предобработка и Масштабирование (как методы без учителя):**

* Многие алгоритмы МО (особенно SVM, нейронные сети, методы на основе расстояний) чувствительны к масштабу признаков.
* Масштабирование преобразует признаки так, чтобы они имели схожий диапазон или распределение. Это неконтролируемый процесс, так как он использует только сами значения признаков (X), без меток (y).
* Основные методы масштабирования в scikit-learn:

**StandardScaler**: Приводит данные к нулевому среднему и единичной дисперсии (Z-оценка). Не гарантирует конкретных min/max. Чувствителен к выбросам.

**RobustScaler**: Использует медиану и квартили. Устойчив к выбросам. Также приводит к схожему масштабу, но не обязательно к [0, 1] или среднему 0.

**MinMaxScaler**: Масштабирует данные в заданный диапазон, по умолчанию [0, 1]. Гарантирует, что все признаки будут в этих границах (на обучающих данных). Чувствителен к выбросам.

**Normalizer**: Масштабирует каждый образец (строку) индивидуально так, чтобы его вектор признаков имел единичную длину (L2-норму по умолчанию). Фокусируется на направлении вектора, а не на абсолютных значениях. Используется реже, в основном для текстовых данных или когда важны углы между векторами.

**Процесс применения Преобразований (в scikit-learn):**

* Инициализация: Создать экземпляр трансформера (например, scaler = MinMaxScaler()).
* Обучение (fit): Вычислить необходимые статистики на обучающих данных (scaler.fit(X\_train)). Например, для MinMaxScaler это min и max каждого признака. fit никогда не используется на тестовых данных!
* Преобразование (transform): Применить вычисленное преобразование к данным (X\_train\_scaled = scaler.transform(X\_train), X\_test\_scaled = scaler.transform(X\_test)). Важно: к тестовым данным применяется то же самое преобразование, обученное на обучающих данных!
* Комбинированный метод (fit\_transform): Выполняет fit и transform за один шаг. Удобно применять к обучающим данным (X\_train\_scaled = scaler.fit\_transform(X\_train)). Никогда не использовать fit\_transform на тестовых данных!

**Почему важно масштабировать Train и Test одинаково?**

* Модель обучается на данных с определенной структурой и масштабом.
* Чтобы модель корректно работала на новых (тестовых) данных, эти данные должны быть приведены к тому же масштабу, который модель "видела" при обучении.
* Масштабирование тестовых данных отдельно (с использованием их собственных min/max или среднего/ст.откл.) искажает их структуру относительно обучающих данных, что приведет к неверным предсказаниям.

**Влияние масштабирования на Обучение с учителем:**

* Как показано на примере с SVM, правильное масштабирование может кардинально улучшить качество модели (точность на тестовом наборе выросла с 0.63 до 0.97 для MinMaxScaler и до 0.96 для StandardScaler).
* Для моделей, нечувствительных к масштабу (например, деревья решений, случайный лес), масштабирование не требуется и не даст эффекта.

**Объяснение графиков (Рисунки из Лабораторной работы №29):**

**Рис. 1:**Сравнение 4 методов масштабирования (StandardScaler, RobustScaler, MinMaxScaler, Normalizer) на синтетических 2D данных.

* Original Data: Исходные данные. Видно, что признак 0 имеет диапазон [~10, ~15], а признак 1 - [~1, ~9]. Масштабы разные.
* StandardScaler: Данные центрированы около (0, 0), разброс по обеим осям примерно одинаков (дисперсия 1). Форма облака сохранена.
* RobustScaler: Похоже на StandardScaler, но может немного отличаться центр и разброс, если были выбросы (здесь их нет).
* MinMaxScaler: Все данные "вписаны" в квадрат [0, 1] x [0, 1]. Форма облака может немного исказиться.
* Normalizer: Все точки "спроецированы" на единичную окружность (или ее часть). Видно, что точки теперь лежат на дуге. Этот метод сильно меняет структуру данных.

**Рис. 2:**Иллюстрация правильного и неправильного масштабирования обучающего и тестового наборов.

* Исходные данные: Обучающие (кружки) и тестовые (треугольники) наборы данных make\_blobs.
* Масштабированные данные (правильно): Оба набора масштабированы с помощью MinMaxScaler, обученного только на обучающем наборе. Тестовые точки сохранили свое положение относительно обучающих, хотя некоторые вышли за пределы квадрата [0, 1] (это нормально).
* Неправильно масштабированные данные: Обучающий набор масштабирован своим скейлером, а тестовый - своим собственным скейлером. В результате и обучающий, и тестовый наборы идеально вписаны в [0, 1], но их взаимное расположение искажено. Тестовые точки "сжались" и "сдвинулись" относительно обучающих. Модель, обученная на правильно масштабированных данных, скорее всего, плохо сработает на этих неправильно масштабированных тестовых данных.

# Лаб 21 (30) (9 рисунок чет не то)

**PCA** (Principal Component Analysis) - это метод обучения без учителя для преобразования данных, в первую очередь для снижения размерности.

**Основная идея:** найти новые оси (главные компоненты) в данных, вдоль которых дисперсия (разброс) максимальна.

PCA вращает данные так, чтобы новые оси (главные компоненты) стали некоррелированными. Первая главная компонента (ГК1) соответствует направлению максимальной дисперсии, ГК2 - направлению максимальной дисперсии среди всех направлений, ортогональных ГК1, и т.д.

Компоненты ортогональны друг другу

Перед вращением PCA центрирует данные (вычитает среднее значение по каждому признаку), так что преобразованные данные имеют нулевое среднее.

**Как работает PCA?**

* Находит собственный базис ковариационной матрицы данных. Собственные векторы этой матрицы задают направления главных компонент, а соответствующие им собственные значения показывают дисперсию данных вдоль этих направлений.
* Компоненты сортируются по убыванию собственных значений (т.е. по убыванию объясненной дисперсии).

**Применения PCA:**

**Снижение размерности:**Оставить только первые несколько (k) главных компонент, которые объясняют бОльшую часть дисперсии данных. Это позволяет уменьшить количество признаков, сохранив при этом основную информацию. Количество компонент (k) обычно выбирается пользователем (n\_components в scikit-learn).

**Визуализация:**Снизить размерность данных до 2 или 3 компонент, чтобы отобразить их на диаграмме рассеяния и попытаться увидеть структуру (кластеры, зависимости).

**Выделение признаков (Feature Extraction):**Главные компоненты являются линейными комбинациями исходных признаков. Иногда эти комбинации могут иметь содержательную интерпретацию или быть более полезными для последующих алгоритмов, чем исходные признаки (пример: Eigenfaces).

**Удаление шума:**Компоненты с очень малой дисперсией часто соответствуют шуму в данных. Отбрасывая их, можно "очистить" данные.

**Декорреляция признаков:** Преобразованные данные (в пространстве ГК) имеют некоррелированные признаки.

**Интерпретация главных компонент:**

* **Каждая главная компонента** - это вектор в исходном пространстве признаков. Координаты этого вектора (коэффициенты) показывают вклад каждого исходного признака в эту компоненту.
* Анализ этих коэффициентов (например, с помощью теплокарты) может помочь понять, какие комбинации исходных признаков важны для объяснения дисперсии. Однако часто эти комбинации сложны и трудно интерпретируемы.
* В примере с раком груди, ГК1 имела примерно одинаковые по знаку коэффициенты для всех признаков (общая корреляция), а ГК2 - разные знаки (противопоставление одних признаков другим).

**Метод "Собственных лиц" (Eigenfaces):**

* Классический пример применения PCA к изображениям лиц.
* Каждое изображение представляется как длинный вектор пикселей.
* PCA применяется к набору таких векторов.
* Найденные главные компоненты (если их снова преобразовать в форму изображения) выглядят как "призрачные" лица - eigenfaces. Они представляют основные направления вариации в лицах (освещение, выражение, общие черты).
* Изображение можно представить как взвешенную сумму этих eigenfaces. Коэффициенты этой суммы - это координаты изображения в новом пространстве главных компонент.
* Использование этих PCA-признаков (коэффициентов) вместо исходных пикселей может значительно улучшить качество классификации лиц (например, с помощью k-NN), так как они лучше отражают схожесть лиц, чем попиксельное сравнение.

**Выбеливание (Whitening):**

* Опция в PCA (whiten=True), которая масштабирует главные компоненты так, чтобы они имели единичную дисперсию.
* Это аналогично применению StandardScaler после PCA-преобразования.
* Может быть полезно для некоторых алгоритмов, которые предполагают, что признаки имеют одинаковую дисперсию (хотя PCA и так делает их некоррелированными).
* Визуально, если исходные данные образовывали эллипс, после PCA с выбеливанием они будут образовывать круг (в 2D).

**Обратное преобразование (inverse\_transform):**

* PCA позволяет реконструировать (приближенно) исходные данные из данных, спроецированных на главные компоненты.
* Если использовать все главные компоненты, реконструкция будет точной (с учетом числовых погрешностей).
* Если использовать только первые k компонент, реконструкция будет приближенной. Чем больше компонент используется, тем точнее реконструкция.
* Пример: Реконструкция лиц с разным числом компонент показывает, что первые компоненты кодируют общие черты (освещение, ориентация), а последующие добавляют детали.

**Важность масштабирования ДО PCA:**

PCA очень чувствителен к масштабу исходных признаков. Он ищет направления максимальной дисперсии. Если один признак имеет гораздо большую дисперсию (из-за единиц измерения, например), чем другие, то PCA "сконцентрируется" на этом признаке, и результат будет неинформативным.

Критически важно масштабировать данные (например, с помощью StandardScaler) перед применением PCA.

**PCA как неконтролируемый метод:**

Важно помнить, что PCA не использует информацию о метках классов (y) при поиске главных компонент. Он анализирует только структуру самих данных (X).

Поэтому нет гарантии, что направления максимальной дисперсии будут лучшими для разделения классов. Хотя часто это так (как в примере с раком груди). В примере с лицами первые две компоненты не смогли хорошо разделить классы.

**Объяснение графиков (Рисунки из Лабораторной работы №30):**

**Рис. 1:**Иллюстрирует шаги PCA на синтетических 2D данных.

* Вверху слева: Исходные данные, видна корреляция между признаками. Выделена первая главная компонента (Component 1) - направление максимального разброса, и вторая (Component 2) - ортогональная ей.
* Вверху справа: Данные после поворота так, чтобы главные компоненты совпали с осями координат. Данные теперь не коррелированы.
* Внизу слева: Данные после снижения размерности до 1 компоненты. Точки спроецированы на направление первой главной компоненты. Информация о второй компоненте потеряна.
* Внизу справа: Данные после реконструкции из одной главной компоненты. Они лежат на прямой, соответствующей первой ГК в исходном пространстве. Показывает, какая информация была сохранена (разброс вдоль ГК1), а какая потеряна.

**Рис. 2:**Гистограммы распределения каждого из 30 признаков датасета Breast Cancer для двух классов (злокачественная/доброкачественная). Позволяет визуально оценить, какие признаки могут быть полезны для разделения классов. Например, "worst concave points" выглядит информативно, а "smoothness error" - нет. Мотивирует применение методов снижения размерности, т.к. анализировать 30 признаков попарно сложно.

**Рис. 3:**Диаграмма рассеяния датасета Breast Cancer в пространстве первых двух главных компонент. Данные были предварительно масштабированы, затем применен PCA с n\_components=2. Видно, что классы (malignant/benign) в этом новом 2D пространстве разделяются гораздо лучше, чем на любой паре исходных признаков. Это показывает эффективность PCA для визуализации и потенциально для улучшения классификации.

**Рис. 4:** Теплокарта, показывающая состав первых двух главных компонент. Строки - компоненты (1-я, 2-я), столбцы - исходные 30 признаков. Цвет показывает вес (коэффициент) каждого исходного признака в линейной комбинации, образующей главную компоненту. Помогает понять (хотя и сложно), какие признаки вносят основной вклад в направления максимальной дисперсии.

**Рис. 5:** Примеры изображений лиц из датасета Labeled Faces in the Wild (LFW). Демонстрирует тип данных, к которым будет применяться PCA далее.

**Рис. 6:** Иллюстрация опции PCA Whitening. Исходные данные (эллипс) сначала поворачиваются PCA (разброс вдоль осей), а затем масштабируются так, чтобы дисперсия вдоль каждой оси стала равной 1 (эллипс превращается в круг).

**Рис. 7:** Визуализация первых 15 главных компонент, найденных PCA для датасета лиц LFW. Каждая компонента представлена как изображение того же размера, что и исходные лица. Это и есть "собственные лица" (Eigenfaces). Видно, что первые компоненты кодируют глобальные характеристики (освещение, общие черты), а последующие - более мелкие детали.

**Рис. 8:**Схематическое изображение реконструкции лица (~X) как взвешенной суммы главных компонент (собственных лиц X0\*, X1\*, ...) с коэффициентами (x0, x1, ...), полученными после PCA-преобразования.

**Рис. 9:** Результаты реконструкции трех лиц с использованием разного количества главных компонент (10, 50, 100, 500). Видно, как с увеличением числа компонент изображение становится все более четким и похожим на оригинал.

**Рис. 10:**Диаграмма рассеяния данных LFW (обучающий набор) в пространстве первых двух главных компонент. В отличие от датасета рака (Рис. 3), здесь нет четкого разделения на классы (разные люди). Это показывает, что первые две ГК, хотя и объясняют максимум дисперсии (вероятно, связанной с освещением и позой), недостаточны для идентификации личности.

# Лаб 22 (31)

**Факторизация неотрицательных матриц** (Non-negative Matrix Factorization, NMF):

**Цель**: Как и PCA, используется для выделения полезных характеристик (компонент) и снижения размерности.

**Ключевое отличие от PCA**: NMF требует, чтобы и исходные данные, и найденные компоненты, и коэффициенты разложения были неотрицательными (>= 0).

**Применение**: Особенно полезен для данных, которые представляют собой сумму или наложение нескольких неотрицательных источников (например, аудиозапись с несколькими голосами, изображение как сумма паттернов, счетчики слов в документах). NMF пытается найти эти исходные "составляющие".

**Интерпретируемость**: Из-за неотрицательности компоненты NMF часто более интерпретируемы, чем компоненты PCA (которые могут содержать и положительные, и отрицательные веса, приводя к взаимокомпенсации). Компоненты NMF можно рассматривать как "части" или "прототипы", из которых "собираются" исходные данные.

**Ограничения**: Применим только к неотрицательным данным.

Свойства компонент:

* Компоненты NMF не ортогональны (в отличие от PCA).
* Компоненты NMF не упорядочены по "важности" или объясненной дисперсии (в отличие от PCA). Все компоненты считаются равнозначными.
* Алгоритм использует случайную инициализацию, поэтому результат может немного меняться от запуска к запуску, особенно на сложных данных.

**Пример с лицами:**Компоненты NMF для лиц выглядят как фрагменты или прототипы лиц (более похожи на реальные части, чем Eigenfaces из PCA). Лица, сильно "активирующие" компоненту "лицо вправо", действительно смотрят вправо.

**Пример с сигналами:** NMF успешно разделяет смешанные сигналы на исходные компоненты, в то время как PCA выделяет компоненты по максимальной дисперсии, что не соответствует исходным сигналам.

**Множественное обучение (Manifold Learning)**

Класс алгоритмов, предназначенных в основном для визуализации данных высокой размерности в 2D или 3D.

Идея: Найти низкоразмерное представление ("многообразие"), которое наилучшим образом сохраняет структуру соседства точек из исходного пространства. Точки, близкие в исходном пространстве, должны остаться близкими и в новом, низкоразмерном представлении.

Алгоритмы часто сложнее PCA.

Обычно не поддерживают метод transform для новых данных, т.е. работают только с теми данными, на которых были обучены (используют fit\_transform). Поэтому редко применяются как предобработка для обучения с учителем.

Используются для разведочного анализа данных.

**t-SNE (t-distributed Stochastic Neighbor Embedding):**

Один из самых популярных и эффективных алгоритмов множественного обучения для визуализации.

**Цель**: Найти такое 2D (или 3D) представление, в котором точки, близкие в исходном пространстве, будут с высокой вероятностью близки и в 2D, а далекие точки будут с высокой вероятностью далеки.

**Особенности**: Уделяет большее внимание сохранению локальной структуры (отношений между близкими соседями), чем глобальной структуре (отношений между далекими кластерами).

**Результат**: Часто создает очень четкие, хорошо разделенные кластеры на 2D-графике, даже если в исходном пространстве они перекрывались.

**Параметры**:

* perplexity: Связан с предполагаемым числом соседей точки. Значение по умолчанию (30) часто работает хорошо.
* early\_exaggeration: Контролирует "рассталкивание" кластеров на ранних этапах.
* random\_state: Важен, так как алгоритм использует случайную инициализацию.

**Интерпретация результата:** Хотя кластеры выглядят четко, расстояния между кластерами на t-SNE графике и размеры самих кластеров обычно не несут содержательной информации. Главное - какие точки попали в один кластер.

**Объяснение графиков (Рисунки из Лабораторной работы №31):**

**Рис. 1** (на стр. 2, подпись Рис. 15 в OCR некорректна): Компоненты NMF для синтетических 2D данных

**Слева**: NMF с 2 компонентами. Компоненты (черные стрелки) указывают на "края" облака данных, т.к. все точки можно представить как неотрицательную комбинацию этих двух направлений.

**Справа**: NMF с 1 компонентой. Компонента указывает на "среднее" направление данных, т.к. это лучшее одиночное неотрицательное направление для объяснения данных. Показывает, что NMF работает иначе, чем PCA при снижении размерности.

**Рис. 2:**Реконструкция лиц с помощью NMF с разным числом компонент (10, 50, 100, 500). Аналогично Рис. 9 из ЛР30 (для PCA). Качество реконструкции улучшается с ростом числа компонент. Визуально может быть немного хуже, чем у PCA при том же числе компонент, т.к. цель NMF - не оптимальная реконструкция, а выделение интерпретируемых частей.

**Рис. 3:**Визуализация первых 15 компонент NMF, извлеченных из датасета лиц LFW. В отличие от Eigenfaces (PCA), эти компоненты неотрицательны и больше похожи на фрагменты или прототипы лиц (глаза, носы, рты, контуры при разном освещении или повороте).

**Рис. 4:**10 лиц из обучающего набора, имеющих максимальный коэффициент для 3-й компоненты NMF. Видно, что это лица, повернутые вправо, что соответствует виду самой 3-й компоненты на Рис. 3.

**Рис. 5:** 10 лиц, имеющих максимальный коэффициент для 7-й компоненты NMF. Это лица, смотрящие влево/слегка вверх, что также соответствует виду 7-й компоненты. Подтверждает интерпретируемость компонент NMF.

**Рис. 6:**Три исходных ("чистых") сигнала, которые будут смешаны.

**Рис. 7:**Сравнение восстановления сигналов с помощью NMF и PCA.

Верхний график: Первые 3 "измерения" (смешанные сигналы).

Второй сверху: Исходные сигналы (для сравнения).

Третий сверху: Сигналы, восстановленные NMF. Видно, что NMF успешно разделил смесь и нашел сигналы, очень похожие на исходные (с точностью до масштаба и порядка).

Нижний график: Сигналы (компоненты), найденные PCA. PCA не смог выделить исходные сигналы, т.к. он ищет ортогональные направления максимальной дисперсии, что не соответствует структуре смешения.

**Рис. 8:** Примеры изображений рукописных цифр из датасета digits.

**Рис. 9:**Визуализация датасета digits с помощью PCA (первые 2 компоненты). Точки (цифры) раскрашены по классам. Видно некоторое разделение (0, 6, 4 отделяются), но большинство классов сильно перекрываются.

**Рис. 10:**Визуализация датасета digits с помощью t-SNE. Результат гораздо лучше: классы цифр образуют отчетливые, хорошо разделенные группы. Это демонстрирует силу t-SNE для визуализации и поиска кластерной структуры в данных.