

Экзаменационные вопросы по курсу "Основы машинного обучения"

1. Что такое машинное обучение? Кто такой Data Scientist? Как машинное обучение и наука о данных связаны с искусственным интеллектом?

Машинное обучение (мл) – класс методов искусственного интеллекта (ии), характерной чертой которых является не прямое решение задачи, а обучение (выявление закономерностей) за счёт применения решений множества сходных задач. Для построения таких методов используются средства математической статистики, численных методов, математического анализа, методов оптимизации, теории вероятностей и программирования.

Искусственный интеллект (ии) – область исследований в области компьютерных наук, которая разрабатывает и изучает методы, позволяющие машинам воспринимать окружающую среду и использовать обучение и интеллект для выполнения разных задач. К таким задачам относятся восприятие и распознавание образов (компьютерное зрение), принятие решений, обработка естественного языка, и обучение на основе опыта.

Data Scientist – это профессионал, специализирующийся на извлечении ценной информации и из данных. Он совмещает навыки в математике, статистике, программировании и предметной области, чтобы анализировать сложные данные, строить модели, интерпретировать результаты и формулировать выводы, полезные для принятия решений.

МЛ является подразделом ИИ, который направлен на создание “псевдоинтеллектуальных” алгоритмов, способных решать задачи на основе знаний, полученных при обучении. Также МЛ неразрывно связано с наукой о данных, так как оперирует теми знаниями, которые были получены из набора данных.

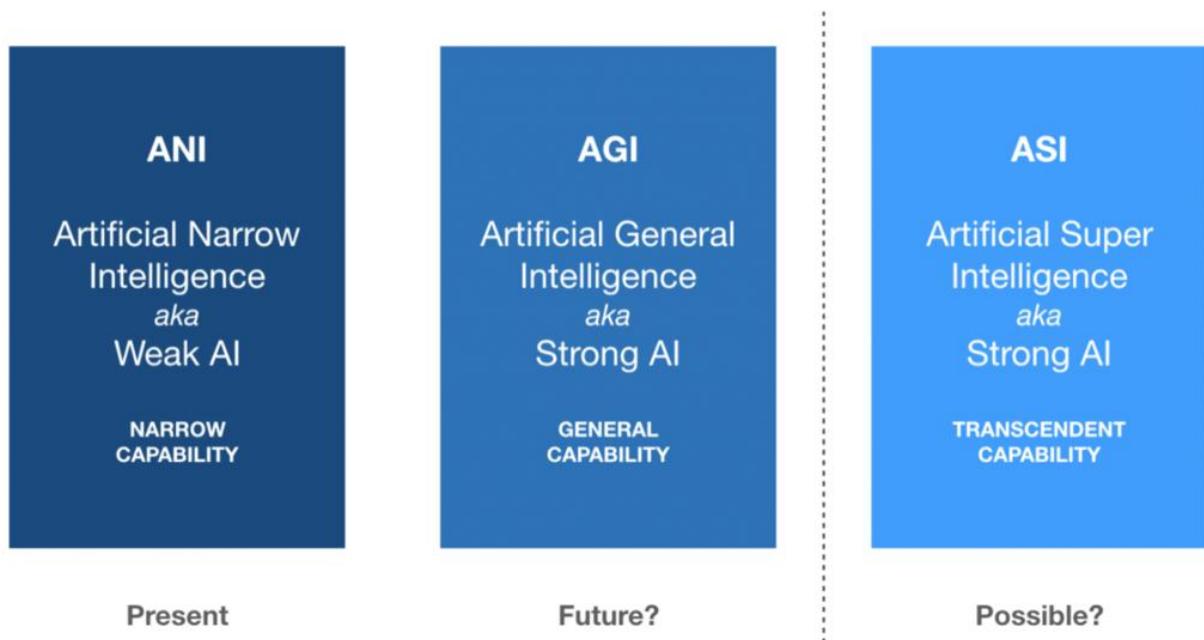
2. Уровни развития искусственного интеллекта (слабый, сильный, ANI, AGI, ASI).

Уровень развития искусственного интеллекта классифицируется по уровням в зависимости от его способностей и степени приближения к человеческому интеллекту.

Слабый ИИ или узкий ИИ (ANI) представляет собой системы, способные решать конкретные, ограниченные задачи, строго в пределах заданной области. Такие системы не обладают сознанием, самосознанием или пониманием контекста за пределами своей задачи. Современные технологии ИИ, включая рекомендательные алгоритмы, голосовых помощников, распознавание изображений, машинный перевод и большинство моделей машинного обучения, относятся к этому уровню.

Сильный ИИ или общий ИИ (AGI) представляет гипотетический этап развития ИИ, на котором система способна выполнять любые интеллектуальные задачи, доступные человеку. Такая система может самостоятельно обучаться, адаптироваться, переносить знания между доменами и демонстрировать способность к обобщению, логическому мышлению и самосознанию.

Сверхинтеллект (ASI) описывает гипотетический уровень развития ИИ, который превосходит человека во всех аспектах интеллектуальной деятельности, включая творчество, социальные навыки, научное мышление и стратегическое планирование.



3. История развития ИИ, МО и глубокого обучения.

1940–1950-е годы – формирование теоретических основ. Появление логики, кибернетики, формализации понятий алгоритма и машины (включая работу Алана Тьюринга “Вычислительные машины и разум”, в которой предложил знаменитый тест Тьюринга — критерий, позволяющий определить, может ли машина мыслить), стало

фундаментом будущего ИИ. В 1956 году на конференции в Дартмуте был официально введён термин "искусственный интеллект", что положило начало становлению дисциплины как отдельного направления.

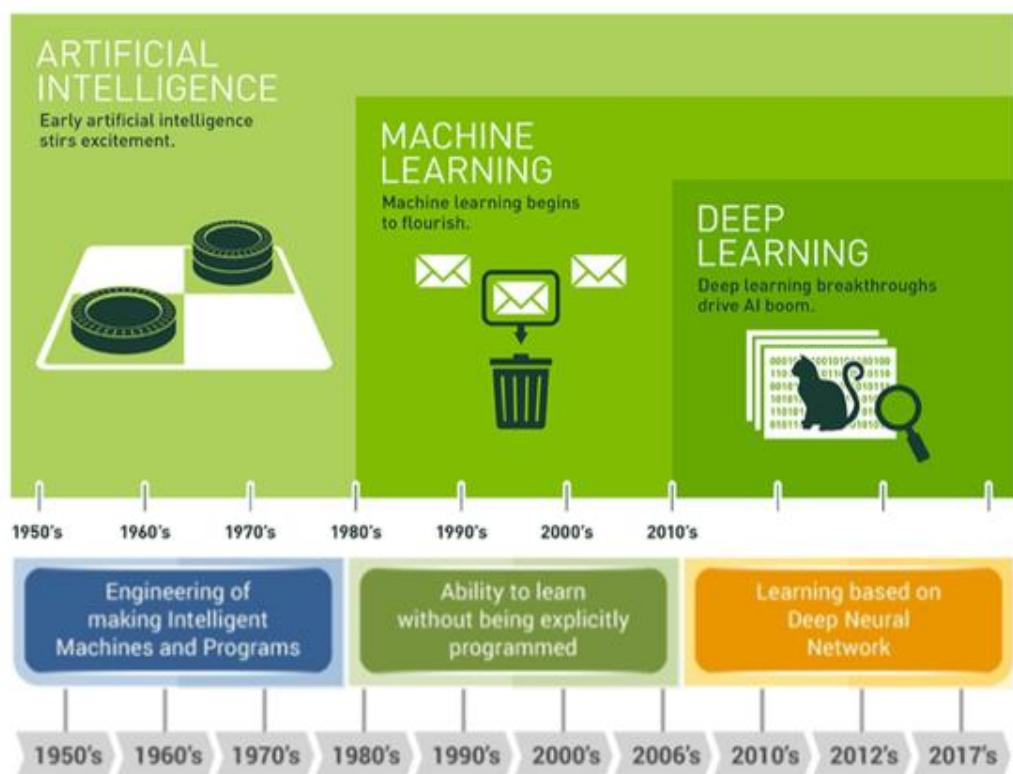
В 1957 году американский психолог Фрэнк Розенблattt (1928–1971) создал первую обучаемую нейросеть — персепtron. Это была простейшая модель, способная распознавать образы и делать выводы на основе входных данных. Создавались системы, использующие жёстко заданные правила и дерево решений. Однако символические системы плохо справлялись с неопределенностью и вариативностью реального мира, что вскоре привело к снижению интереса, известному как первый “кризис ИИ” (Зима ИИ).

Некоторые ученые продолжали исследования, пытаясь преодолеть существующие ограничения. Стали популярны подходы, основанные на символической логике и экспертных системах. Это были программы, в которых знания задавались вручную в виде правил и логических связей. Они не относились напрямую к обучающимся системам, но повлияли на дальнейшее развитие ИИ. Высокая стоимость разработки, проблемы масштабирования и ограниченность обобщающих способностей вновь привели к разочарованию, и второй «кризис ИИ» произошёл в конце 1980-х годов.

Развитие машинного обучения как отдельного направления началось с осознания необходимости автоматического извлечения знаний из данных, без явного программирования всех возможных сценариев. В 1990-е годы на передний план выходят методы статистического обучения, включая логистическую регрессию, метод опорных векторов (SVM), деревья решений и ансамбли моделей. Важным фактором стала цифровизация данных и бурный рост объёмов информации, накопленной в интернете и информационных системах. Это создало основу для интенсивного развития подходов, ориентированных на данные, и перехода от эвристического программирования к моделям, способным обучаться.

С начала 2000-х годов развитие ИИ тесно связано с прогрессом в вычислительной технике (GPU, распределённые вычисления), доступностью больших данных и совершенствованием алгоритмов обучения. Эти условия привели к возрождению интереса к нейронным сетям и к формированию парадигмы глубокого обучения, отличительной чертой которого стало использование нейросетей с большим числом слоёв и параметров.

Прорывом стало выступление модели AlexNet на конкурсе ImageNet в 2012 году, где была достигнута значительная точность в задаче классификации изображений. Архитектура CNN (сверточные нейронные сети) стала стандартом в задачах компьютерного зрения. Далее появились RNN и LSTM для работы с последовательностями, GAN для генеративного моделирования, а также трансформеры, начиная с архитектуры Attention Is All You Need (2017), обеспечившие основу современных моделей обработки естественного языка, таких как BERT, GPT и их производные.



4. В каких областях применяется машинное обучение? Приведите примеры решения прикладных задач с помощью МО.

Существует множество задач, в которых может применяться МЛ:

- Автоматический перевод языков;
- Выставление медицинских диагнозов;
- Торговля на фондовом рынке;
- Выявление онлайн-мошенничества;
- Виртуальный ассистент;
- Фильтрация спам-писем;

- Машины с автоуправлением;
- Алгоритмы рекомендации продуктов;
- Прогнозирование трафика;
- Распознавание речи/изображений;
- И тп.

5. Постановка задачи обучения на примерах.

Классические методы МО обычно используют для решения ключевых типов задач, каждая из которых имеет свои особенности и применения: регрессия, ранжирование, классификация, кластеризация, уменьшение размерности, выявление аномалий.

- Регрессия. Она представляет собой тип задачи, где цель состоит в предсказании непрерывного числового значения на основе входных данных. (Предсказание цены дома по таким признакам как площадь, количество комнат, район и год постройки);
- Ранжирование. Это тип задачи, где цель состоит в упорядочении объектов по их релевантности или важности относительно определенного критерия (запроса). (Рекламные или рекомендательные системы);
- Классификация. Это тип задачи, где цель состоит в определении принадлежности объекта к определенному классу объектов или категории на основе его характеристик (признаков). (Классификация электронных писем на "спам" и "не спам" по признакам, таким как наличие определенных слов, длина письма и адрес отправителя);
- Кластеризация. Цель кластеризации состоит в том, чтобы сгруппировать похожие объекты в кластеры на основе их признаков, без заранее известных меток классов или значений для предсказания. (группировка клиентов интернет-магазина по их покупкам, демографическим характеристикам и поведению на сайте, чтобы персонализировать их предпочтения);
- Уменьшение размерности. Это процесс сокращения количества признаков в данных, сохраняя при этом максимально возможное количество информации.
- Выявление аномалий. Это задача, целью которой является идентификация данных, которые существенно отличаются от остальных данных и не следуют ожидаемым закономерностям.

В общем случае целью всех типов задач в МЛ является выведение зависимости в данных:

- X – множество *объектов*
- Y – множество *ответов* (предсказаний, оценок, прогнозов)
- $\varphi(x), \varphi: X \rightarrow Y$ – неизвестная зависимость (target function)

Дано:

- $\{x_1, \dots, x_\ell\} \subset X$ – обучающая выборка (training sample)
- $y_i = \varphi(x_i), i = 1, \dots, \ell$ – известные ответы

Найти:

- $g(x, \theta), g: X \times \Theta \rightarrow Y$ – алгоритм, функция принятия решений или параметрическая модель, приближающая φ на всей выборке X
- $\theta \in \Theta$ – вектор параметров модели, такой, что $g(x, \theta) \approx \varphi(x)$

6. Описание объектов и ответов. Типы задач машинного обучения.

Описание объектов:

$f_j: X \rightarrow D_j, j = 1, \dots, n$ – признаки объектов (features)

Типы скалярных признаков:

- $D_j = \{0,1\}$ – бинарный признак f_j ;
- $|D_j| < \infty$ – номинальный признак f_j ;
- $|D_j| < \infty$, D_j упорядочено – порядковый признак f_j ;
- $D_j = \mathbb{R}$ – количественный признак f_j : интервал или число.

Вектор $(f_1(x), \dots, f_n(x))$ – признаковое описание объекта x .

Матрица признаков: $F = \|f_j(x_i)\|_{\ell \times n} = \begin{pmatrix} f_1(x_1) & \dots & f_n(x_1) \\ \vdots & \ddots & \vdots \\ f_1(x_\ell) & \dots & f_n(x_\ell) \end{pmatrix}$

Описание ответов:

Задача классификации:

- $Y = \{-1, +1\}$ – бинарная классификация (два класса);
- $Y = \{1, \dots, M\}$ – классификация между M не пересекающимися классами;
- $Y = \{0,1\}^M$ – M классов, которые могут пересекаться.

Задача регрессии:

- $Y = \mathbb{R}$ or $Y = \mathbb{R}^m$.

Задача ранжирования:

- Y – конечное отсортированное множество.

7. Обучение с учителем, предсказательные модели. Приведите не менее 4-х примеров описания прикладных задач.

Обучение с учителем представляет собой класс методов машинного обучения, при которых алгоритм обучается на размеченных данных, то есть каждому входному примеру соответствует заранее известный правильный ответ. Целью обучения с учителем является построение модели, способной обобщать закономерности из обучающей выборки и корректно предсказывать ответы на новых, ранее не встречавшихся данных.

Обучение по прецедентам является классическим подходом к МЛ, где под прецедентом подразумевается множество наблюдаемых случаев (обучающая выборка).

В контексте прикладных задач обучение с учителем используется для решения задач классификации (определение класса объекта) и регрессии (предсказание непрерывного значения). Центральной задачей является восстановление функциональной зависимости между входными признаками и выходной переменной на основании эмпирических данных.

Предсказательное моделирование – процесс создания модели, способной давать точные прогнозы на основе признаков, которые не встречаются в обучающем наборе данных.

Описание прикладных задач:

- Диагностика заболеваний по медицинским показателям. Данные содержат информацию о пациенте (возраст, уровень холестерина, давление,

результаты анализов), а целевая переменная — наличие или отсутствие заболевания (Классификация).

- Прогнозирование спроса на товар. Обучающая выборка включает информацию о продажах в предыдущие периоды, сезонные показатели, рекламу и экономические индикаторы. Целевая переменная — количество продаж в следующем месяце (Регрессия).
- Распознавание рукописных цифр. Используются изображения цифр, представленные в виде массива пикселей (входные признаки), а целевая переменная — соответствующая цифра (0–9) (Множественная классификация).
- Предсказание времени отказа промышленного оборудования. Имеются данные с датчиков (температура, вибрация, давление) и информация о фактическом времени выхода из строя оборудования. Модель обучается предсказывать остаточный срок службы (Регрессия, а именно аппроксимация функции на основе имеющихся данных).

8. Алгоритм обучения. Сведение задачи обучения к задаче оптимизации.

Процесс обучения с учителем состоит из двух этапов:

- **Обучение** (train):

Алгоритм обучения (learning algorithm) μ : $(X \times Y)^\ell \rightarrow \Theta$ по выборке $X^\ell = (x_i, y_i)_{i=1}^\ell$ строит функцию $g(x, \theta)$, оценивая (оптимизируя) параметры модели $\theta \in \Theta$.

- **Применение** (test):

Функция $g(x, \theta)$ для новых объектов x'_i выдает ответы $g(x'_i, \theta)$.

9. Оценивание моделей. Эмпирический риск и функция потерь.

Оценка моделей МЛ осуществляется через вычисление численных характеристик.

Эмпирический риск

- Нельзя заранее достоверно узнать, на сколько хорошо алгоритм g покажет себя на практике («риск»), поскольку неизвестен истинный закон распределения данных $P(x, y)$.
- Оценить и улучшить работу алгоритма g можно на заранее известной ограниченной обучающей выборке (закон больших чисел).
- **Эмпирический риск** — способ оценки качества работы алгоритма g на всей обучающей выборке X^ℓ .
- Функционал эмпирического риска:

$$Q(g, X^\ell) = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathcal{L}(g, x_i)$$

Функция потерь $\mathcal{L}(g, x)$: для заданного объекта $x \in X$ вычисляет величину ошибки алгоритма (функции) $g \in A$ на этом объекте.

Ошибка тем больше, чем сильнее $g(x, \theta)$ отклоняется от правильного ответа $\varphi(x)$.

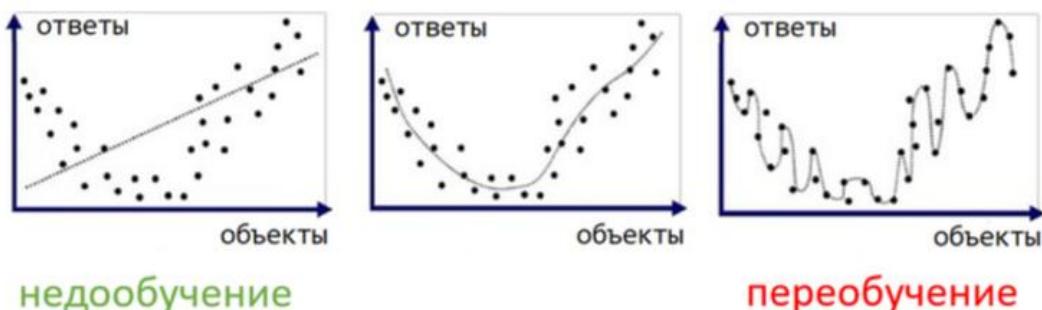
Функция потерь для задач классификации:

- $\mathcal{L}(g, x) = [g(x, \theta) \neq \varphi(x)]$ — индикатор ошибки.

Функция потерь для задач регрессии:

- $\mathcal{L}(g, x) = |g(x, \theta) - \varphi(x)|$ — абсолютное значение ошибки;
- $\mathcal{L}(g, x) = (g(x, \theta) - \varphi(x))^2$ — квадратичная ошибка.

10. Что такое переобучение (overfitting) и недообучение (underfitting)? Как их можно избежать?



Недообучение (underfitting) — это ситуация, при которой модель не способна выявить даже основные закономерности в данных. Она демонстрирует низкое качество как на обучающем, так и на тестовом наборе. Причина недообучения — недостаточная

сложность модели, некорректный выбор признаков, слишком сильная регуляризация или малая продолжительность обучения.

Переобучение (overfitting) — это ситуация, при которой модель слишком точно подстраивается под обучающую выборку, включая её шумы и случайные колебания, что приводит к плохому качеству на новых (тестовых) данных. Такая модель демонстрирует высокую точность на обучающем наборе и низкую — на проверочном или тестовом. Основная причина переобучения — чрезмерная сложность модели, например, слишком большое количество параметров.

Избавится от переобучения нельзя, но его можно уменьшить:

- Увеличивая объём обучающих данных;
- Накладывать ограничения на вектор параметров (регуляризация);
- Выбирать более подходящую модель по оценкам обобщающей способности.

11. Одномерная и многомерная линейная регрессия.

Линейная регрессия представляет собой один из базовых методов машинного обучения, предназначенный для моделирования зависимости между входными переменными (признаками) и выходной переменной (целевой функцией). Метод основывается на предположении, что между переменными существует линейная зависимость, которая может быть выражена в виде алгебраического уравнения. Существуют два основных типа линейной регрессии — одномерная (или простая) и многомерная (или множественная), различающиеся количеством признаков.

Одномерная линейная регрессия имеет вид:

$$y = \theta_0 + \theta^* x$$

где θ_0 — свободный член или смещение, θ — коэффициент наклона или вес, x — признак.

Модель многомерной линейной регрессии можно представить так же, но вместо веса и признака будет вектор весов и вектор признаков.

- $f_1(x), \dots, f_n(x)$ – числовые признаки;
- Модель многомерной линейной регрессии:

$$g(x, \theta) = \sum_{j=1}^n \theta_j f_j(x), \theta \in \mathbb{R}^n$$

Задача обучения модели заключается в нахождении такого значения или вектора значений в общем случае, чтобы минимизировать значение функции ошибок между реальными и предсказанными значениями. Метод наименьших квадратов (МНК) является стандартным методом нахождения весов:

- X – объекты (часто \mathbb{R}^n); Y – ответы (часто \mathbb{R} , реже \mathbb{R}^m);
 $X^\ell = (x_i, y_i)_{i=1}^\ell$ – обучающая выборка;
 $y_i = \varphi(x_i)$, $\varphi: X \rightarrow Y$ – неизвестная зависимость.
- $a(x) = g(x, \theta)$ – модель зависимости,
 $\theta \in \mathbb{R}^p$ – вектор параметров модели.
- Метод наименьших квадратов (МНК):

$$Q(\theta, X^\ell) = \sum_{i=1}^\ell (g(x_i, \theta) - y_i)^2 \rightarrow \min_{\theta}$$

12. Конструирование признаков.

- Использование интуиции для создания новых признаков путем преобразования или комбинирования оригинальных признаков.
 - Пример: предсказание стоимости жилья.
 Признаки: x_1 – площадь квартиры (м. кв.), x_2 – город (категориальный). Модель:

$$g_1(x, \theta) = \theta_2 x_2 + \theta_1 x_1 + \theta_0$$

Добавляем новый признак: $x_3 = x_1 x_2$, чтобы напрямую учесть в модели различия стоимости кв. метра в разных регионах.

$$g_2(x, \theta) = \theta_3 x_3 + \theta_2 x_2 + \theta_1 x_1 + \theta_0$$

- Способы конструирования признаков
 - Полиномиальные признаки: возведение существующих признаков в степень или их комбинирование
 - Агрегация данных: среднее, сумма или медиана по имеющимся признакам
 - Лаги – значения за предыдущие периоды, которые могут влиять на текущие
 - Временные признаки – день недели, месяц или номер квартала, которые учитывают сезонные (периодические) изменения в данных
 - Знания из предметной области

13. Нормализация признаков.

Нормализация признаков — это предварительная обработка данных, заключающаяся в приведении всех признаков к единой шкале. Цель нормализации — устранение различий в масштабах признаков, которые могут негативно повлиять на работу многих алгоритмов машинного обучения.

- Нормирование значений признаков (нормализация средним):

- вычесть математическое ожидание μ_{x_j} признака x_j ;
- поделить на СКО σ_{x_j} ;

$$x_j^* := \frac{x_j - \mu_{x_j}}{\sigma_{x_j}}.$$

- Минимаксная нормализация:

- значения признака приводятся к диапазону $[0,1]$ или $[-1,1]$, например:

$$x_j^* := \frac{x_j - \min x_j}{\max x_j - \min x_j}.$$

- вместо $\min x_j$ в числителе можно использовать среднее μ_{x_i} или медиану.

Когда нужна нормализация данных

- Необходимо стремиться: $-1 \leq x_j \leq 1$ для каждого признака x_j

• Эвристика: нормализация не обязательна, если диапазон признака отличается от $-1 \leq x_j \leq 1$ менее, чем на 2 порядка

- Нормализация не нужна:

$$\begin{aligned} -3 &\leq x_1 \leq 3 \\ -0.3 &\leq x_2 \leq 0.3 \\ 0 &\leq x_3 \leq 3 \\ -2 &\leq x_4 \leq 0.5 \end{aligned}$$

Нормализация обязательна:

$$\begin{aligned} -100 &\leq x_5 \leq 100 \\ -0.001 &\leq x_6 \leq 0.001 \\ 98.6 &\leq x_7 \leq 105 \end{aligned}$$

- Лучше провести нормализацию, чем от нее отказаться

14. Метод наименьших квадратов.

Метод наименьших квадратов (МНК) — это один из фундаментальных методов в статистике и машинном обучении, применяемый для нахождения параметров модели линейной регрессии, минимизируя сумму квадратов отклонений предсказанных значений от фактических наблюдаемых. МНК является только условием нахождения вектора

параметров модели, решением задачи оптимизации являются методы, например, градиентного спуска или метода решения нормальных уравнений.

- X – объекты (часто \mathbb{R}^n); Y – ответы (часто \mathbb{R} , реже \mathbb{R}^m);

$X^\ell = (x_i, y_i)_{i=1}^\ell$ – обучающая выборка;

$y_i = \varphi(x_i)$, $\varphi: X \rightarrow Y$ – неизвестная зависимость.

- $a(x) = g(x, \theta)$ – модель зависимости,

$\theta \in \mathbb{R}^p$ – вектор параметров модели.

- Метод наименьших квадратов (МНК):

$$Q(\theta, X^\ell) = \sum_{i=1}^{\ell} (g(x_i, \theta) - y_i)^2 \rightarrow \min_{\theta}$$

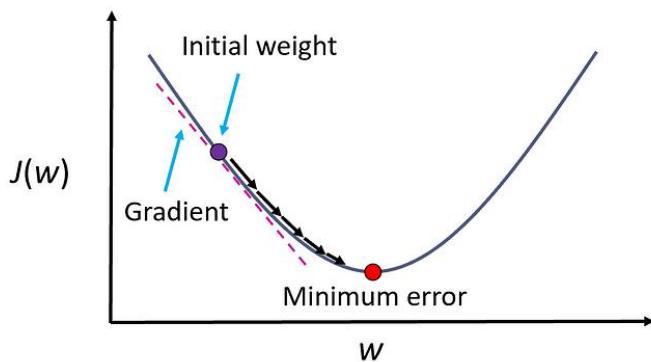
15. Алгоритм градиентного спуска.

Градиентный спуск (Gradient Descent) – итеративный метод оптимизации решения задачи МНК. Его идея заключается в последовательном обновлении параметров модели в направлении антиградиента функции потерь, то есть в направлении наискорейшего убывания функции. Градиент — это вектор частных производных функции по её параметрам, который указывает в сторону наибольшего возрастания функции. Соответственно, движение в противоположную сторону уменьшает значение функции.

На каждой итерации алгоритма происходит вычисление значения функции потерь (или функции стоимости), после чего вектор весов изменяется на один шаг в сторону минимизации функции потерь. Шаг – гиперпараметр, который выбирает разработчик, он называется скоростью обучения (learning rate). Пусть дана функция потерь $J(\theta)$, где θ – вектор весов модели, тогда каждую итерацию алгоритма можно описать формулой:

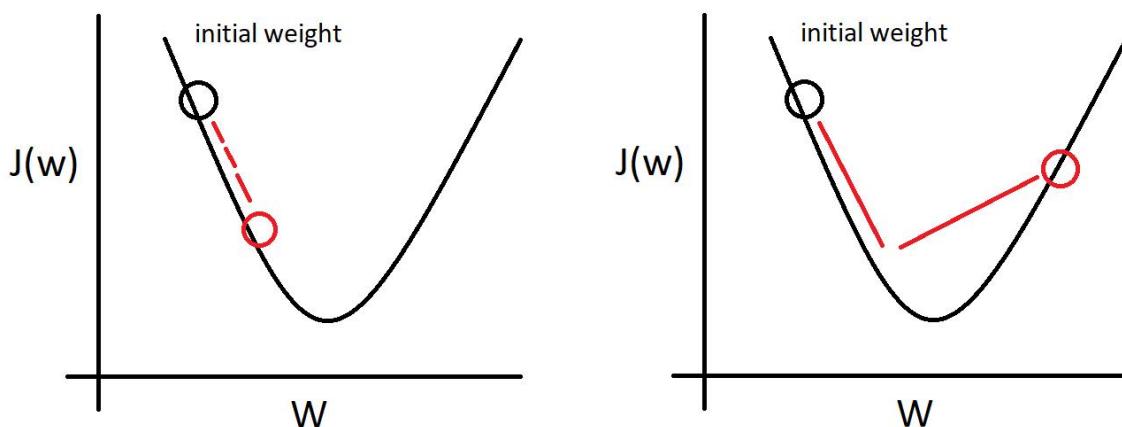
$$\theta^{(i+1)} = \theta^{(i)} - c * J(\theta^{(i)})$$

где $\theta^{(i)}$ – значение весов на i -ой итерации, c – скорость обучения, $J(\theta^{(i)})$ – градиент функции потерь.



Красная точка на графике – локальная точка минимума функции потерь (градиентный спуск не гарантирует нахождение глобального минимума). На первой итерации алгоритма инициализируются веса (точка *initial weight*), далее вычисляется антиградиент, который указывает в сторону ближайшего минимума, стрелки на графике — значения, на которые изменится вектор весов на каждой итерации, каждый такой шаг по длине равен скорости обучения. Для каждого значения функции стоимости есть своё значение весов, поэтому значение весов в точке минимума функции будет считаться оптимальным по завершению работы алгоритма. На графике представлен пример с одним параметром, однако этот пример можно экстраполировать в n -мерное пространство.

Важным моментом в данном алгоритме является выбор скорости обучения, так как при маленькой скорости обучения алгоритм может не дойти до точки минимума функции, а при большом – перескочить эту точку.



16. Стохастический градиентный спуск.

Стохастический градиентный спуск (SGD, Stochastic Gradient Descent) представляет собой модификацию классического градиентного спуска, при которой обновление параметров модели выполняется на основе градиента, вычисленного не по всей

обучающей выборке, а по одному случайному выбранному примеру или по небольшому подмножеству примеров.

- **Проблема:** если обучающая выборка X^ℓ велика ($\ell \gg 0$), то каждый шаг градиентного спуска будет требовать большого количества вычислений, а сам алгоритм будет работать медленно
- **Решение:** Stochastic Gradient Descent, SGD
 - берем по одной паре «объект-ответ» (x_i, y_i) и сразу обновляем вектор θ
 - градиент вычисляем по функции ошибки \mathcal{L} , а не по функционалу качества Q
 - функционал качества оцениваем по приближенной формуле
- **Алгоритм:**
 1. выбрать объект x_i из X^ℓ случайным образом;
 2. вычислить потерю: $\varepsilon_i = \mathcal{L}_i(g, x)$; (напр., $\varepsilon_i = (g(x_i, \theta) - y_i)^2$)
 3. сделать градиентный шаг: $\theta_j = \theta_j - \alpha \frac{\partial}{\partial \theta_j} \mathcal{L}_i(g, x)$;
 4. оценить функционал: $\bar{Q} = \lambda \varepsilon_i + (1 - \lambda) \bar{Q}$, где λ — скорость забывания;
 5. повторять шаги 1-4, пока значение \bar{Q} и/или параметры θ не сойдутся.

Основным преимуществом является значительное снижение вычислительных затрат на одну итерацию, что особенно актуально при работе с большими наборами данных. Однако недостатком является нестабильность траектории сходимости: из-за того, что на каждом шаге используется информация лишь об одной точке, направление обновления может значительно колебаться.

17. Варианты инициализации весов и выбора скорости обучения в алгоритме градиентного спуска.

Цель инициализации весов — задать начальные значения параметров модели, от которых будет производиться итеративное обновление. Варианты инициализации можно разделить на несколько категорий:

1. Нулевая инициализация. Все веса устанавливаются в ноль. Подходит для простых моделей.
2. Случайная инициализация. Весам присваиваются небольшие случайные значения, обычно из нормального или равномерного распределения.
3. Мультистарт (Greed Search). Многократные запуски различных случайных начальных значений.

Скорость обучения определяет шаг изменения параметров на каждой итерации и является гиперпараметром, требующим настройки. Варианты выбора:

1. Фиксированная скорость обучения. Простейший вариант, при котором параметр остаётся постоянным. Требует подбора вручную.
2. Мультистарт или Greed Search. Подразумевает выбор разных значений скорости обучения и многократный запуск с целью выявления лучшего параметра.

18. Использование регуляризации для борьбы с переобучением в алгоритме градиентного спуска.

Использование регуляризации в алгоритме градиентного спуска является одним из ключевых методов борьбы с переобучением (overfitting). Суть регуляризации заключается в добавлении к функции потерь специального штрафа за сложность модели. Это заставляет градиентный спуск не только минимизировать ошибку на обучающей выборке, но и контролировать величину параметров, способствуя лучшей обобщающей способности модели.

L1-регуляризация (лассо) - штраф в виде суммы модулей весов:

$$L = \lambda * |\theta^{(i)}|$$

где λ - гиперпараметр, определяющий силу штрафа. Эта регуляризация приводит к разреживанию модели, то есть некоторые веса становятся строго нулевыми, фактически исключая соответствующие признаки из модели.

L2-регуляризация (ридж) - добавляется к функции потерь штраф в виде суммы квадратов весов:

$$L = \lambda * |\theta^{(i)2}|$$

Эта форма регуляризации способствует уменьшению абсолютных значений весов, делая их ближе к нулю, но не обнуляя полностью.

Методы подбора гиперпараметра лямбда:

1. Скользящий контроль или кросс-валидация. Эмпирический метод (метод, основанный на собственных наблюдениях) определения гиперпараметров. Суть метода заключается в разделении обучающей выборки на K блоков и последовательном обучении модели на K-1 блоках и проверкой на оставшемся

блоке. Этот процесс повторяется K раз, для того чтобы проверить значение гиперпараметра на всех K блоках, итоговое значение параметра усредняется.

2. Стохастическая адаптация. Подразумевает подбор гиперпараметров на основе стохастических методов, работает автоматически.
3. Двухуровневый байесовский вывод. Данный метод учитывает неопределённость значения разных параметров, он позволяет выбрать теоретически обоснованный гиперпараметр, который покрывал как можно больше неопределённостей других параметров.

19. Повышение производительности с помощью векторизации.

Под векторизацией понимается использование операций над векторами и матрицами вместо итеративных циклов. Это позволяет задействовать низкоуровневые оптимизации, встроенные в библиотеки линейной алгебры и аппаратное ускорение, особенно на уровне CPU и GPU.

С точки зрения программной реализации, векторизация означает отказ от явных итераций в пользу операций, применяемых сразу ко всему массиву. Например, прибавление скалярного значения к каждому элементу массива может быть выражено в виде цикла, но гораздо эффективнее — с помощью единственной векторной операции, передаваемой в библиотеку, которая далее выполняется на уровне оптимизированного машинного кода.

Векторизация



- Регрессионная модель:

$$g(\mathbf{x}_i, \theta) = \theta_0 + \sum_{j=1}^n \theta_j x_{i,j} = \sum_{j=0}^n \theta_j x_{i,j}$$

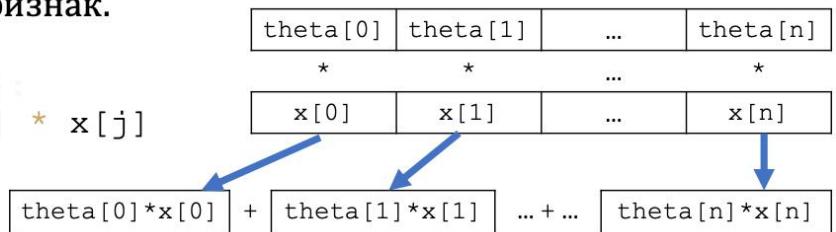
где \mathbf{x}_i – вектор признаков i -го объекта, θ – вектор параметров, $x_{i,0} = 1$ – фиктивный признак.

```
for j in range(0, n):
    g = g + theta[j] * x[j]
```

- В векторном виде:

$$g(\mathbf{x}_i, \theta) = \theta \cdot \mathbf{x}_i \quad \text{быстрее на порядки!}$$

```
import numpy as np
g = np.dot(theta, x)
```



20. Постановка задачи бинарной и многоклассовой классификации.

Бинарная классификация — это частный случай классификации, в котором необходимо определить, принадлежит ли объект к одному из двух возможных классов. Например, при медицинской диагностике — «болен» или «здоров», при проверке электронных писем — «спам» или «не спам», при анализе изображений — «содержит объект» или «не содержит».

Задача обучения классификации

Обучающая выборка: $X^\ell = (x_i, y_i)_{i=1}^\ell, x_i \in \mathbb{R}^n, y_i = \varphi(x_i) \in \{-1, +1\}$

- Модель классификации – линейная с параметром $\theta \in \mathbb{R}^n$:

$$g(x, \theta) = \text{sign}\langle x, \theta \rangle = \text{sign} \sum_{j=1}^n \theta_j f_j(x)$$

- Функция потерь – бинарная или ее аппроксимация:

$$\mathcal{L}(\theta, x) = [g(x, \theta)\varphi(x) < 0] = [\langle x, \theta \rangle \varphi(x) < 0] \leq L(\langle x, \theta \rangle \varphi(x))$$

- Метод обучения – минимизация эмпирического риска:

$$Q(\theta) = \sum_{i=1}^\ell \mathcal{L}(\theta, x_i) = \sum_{i=1}^\ell [\langle x_i, \theta \rangle y_i < 0] \leq \sum_{i=1}^\ell L(\langle x_i, \theta \rangle y_i) \rightarrow \min_{\theta}$$

- Проверка по тестовой выборке $X^k = (\tilde{x}_i, \tilde{y}_i)_{i=1}^k$:

$$\tilde{Q}(\theta) = \frac{1}{k} \sum_{i=1}^k [\langle \tilde{x}_i, \theta \rangle \tilde{y}_i < 0]$$

Многоклассовая классификация представляет собой более общий случай, при котором количество классов превышает два. Подходы к многоклассовой классификации могут быть прямыми, когда модель обучается на все классы одновременно, или основанными на разложении многоклассовой задачи на несколько бинарных, например, через схемы «один против всех» (OvR) или «один против одного» (OvO).

Задача многоклассовой классификации

Обучающая выборка: $X^\ell = (x_i, y_i)_{i=1}^\ell, x_i \in \mathbb{R}^n, y_i = \varphi(x_i) \in Y$

- Модель классификации – линейная, $\theta = (\theta_y : y \in Y)$:

$$g(x, \theta) = \arg \max_{y \in Y} \langle x, \theta_y \rangle$$

- Функция потерь – бинарная или ее аппроксимация:

$$\mathcal{L}(\theta, x) = \sum_{z \neq \varphi(x)} [\langle x, \theta_{\varphi(x)} \rangle < \langle x, \theta_z \rangle] \leq \sum_{z \neq \varphi(x)} L(\langle x, \theta_{\varphi(x)} - \theta_z \rangle)$$

- Метод обучения – минимизация эмпирического риска:

$$Q(\theta) = \sum_{i=1}^\ell \sum_{z \neq y_i} L(\langle x, \theta_{\varphi(x)} - \theta_z \rangle) \rightarrow \min_{\theta}$$

- Проверка по тестовой выборке $X^k = (\tilde{x}_i, \tilde{y}_i)_{i=1}^k$

21. Классификаторы на основе разделяющей поверхности (margin-based).

Классификаторы на основе разделяющей поверхности, также называемые margin-based classifiers, представляют собой класс алгоритмов машинного обучения, задача которых — найти гиперплоскость или более сложную поверхность, разделяющую классы в пространстве признаков с максимальным зазором (margin). Основная идея таких методов заключается не просто в корректной классификации обучающих примеров, но и в том, чтобы делать это с максимально возможной «уверенностью», измеряемой расстоянием от примеров до разделяющей границы.

Преимущества классификаторов с разделяющей поверхностью включают хорошую устойчивость к переобучению, эффективную работу при высокой размерности признаков и интерпретируемость в случае линейной модели. Однако такие модели могут быть чувствительны к выбросам, плохо масштабироваться при очень больших выборках и

требуют внимательного подбора гиперпараметров, таких как коэффициент регуляризации и параметры ядра.

Разделяющие классификаторы (margin-based classifier)

Бинарный классификатор: $g(x, \theta) = \text{sign } h(x, \theta)$, $Y = \{-1, +1\}$

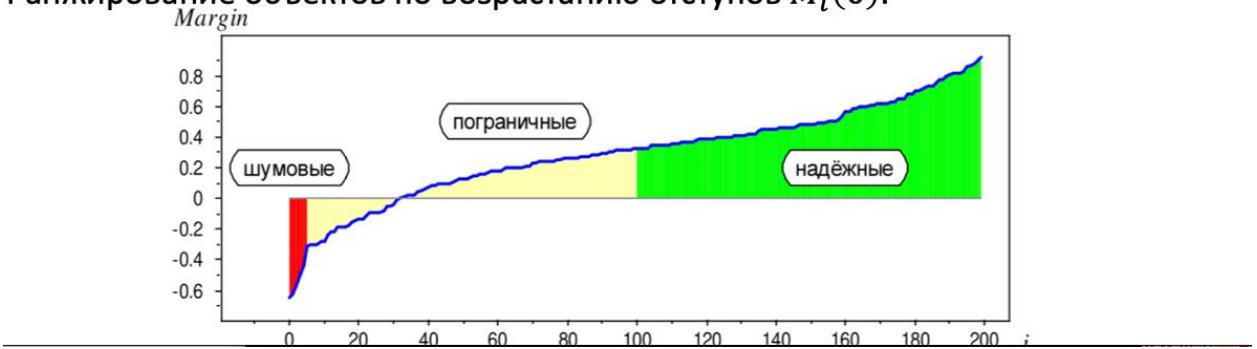
$h(x, \theta)$ – разделяющая (дискриминантная) функция

$x: h(x, \theta) = 0$ – разделяющая поверхность между классами

$M_i(\theta) = h(x_i, \theta)y_i$ – отступ (margin) объекта x_i

$M_i(\theta) < 0 \Leftrightarrow$ алгоритм $g(x, \theta)$ ошибается на x_i

Ранжирование объектов по возрастанию отступов $M_i(\theta)$:



Многоклассовый классификатор:

$$g(x, \theta) = \arg \max_{y \in Y} h_y(x, \theta_y)$$

$h_y(x, \theta_y)$ – дискриминантная функция класса $y \in Y$

$x: h_y(x, \theta_y) = h_z(x, \theta_z)$ – разделяющая поверхность между y, z

$M_{iy}(\theta) = h_{y_i}(x_i, \theta_{y_i}) - h_y(x_i, \theta_y)$ – отступ объекта x_i от класса y

$M_i(\theta) = \min_{y \neq y_i} M_{iy}(\theta)$ – отступ (margin) объекта x_i

$M_i(\theta) < 0 \Leftrightarrow$ алгоритм $g(x, \theta)$ ошибается на x_i

22. Логистическая регрессия.

Логистическая регрессия — это один из базовых методов машинного обучения для решения задачи классификации, основанный на вероятностной модели. В отличие от линейной регрессии, которая предсказывает непрерывное значение, логистическая регрессия используется для предсказания вероятности принадлежности объекта к одному из классов, чаще всего — бинарных: класс 0 или класс 1. Основная идея заключается в

моделировании вероятности принадлежности объекта к положительному классу с помощью логистической функции (сигмоида).

Двухклассовая (бинарная) логистическая регрессия

Линейная модель классификации для двух классов $Y = \{-1, +1\}$:

$$g(x, \theta) = \text{sign}(\theta \cdot x), \quad x, \theta \in \mathbb{R}^n$$

Отступ $M = \langle \theta, x \rangle y$.

Логарифмическая функция потерь:

$$L(M) = \log(1 + e^{-M})$$

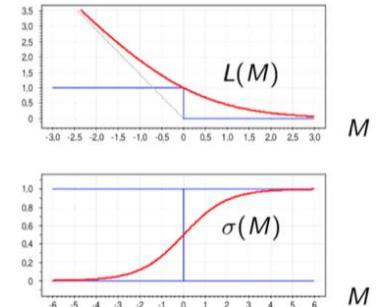
Модель условной вероятности:

$$P(y = 1 | x; \theta) = \sigma(M) = \frac{1}{1 + e^{-M}}$$

где $\sigma(M)$ – сигмоидная (логистическая) функция,
важное свойство: $\sigma(M) + \sigma(-M) = 1$.

Максимизация правдоподобия (logistic loss) с регуляризацией:

$$Q_{MAP}(\theta) = \sum_{i=1}^n \log(1 + \exp(-\langle x, \theta \rangle y_i)) + \frac{\tau}{2} \|\theta\|^2 \rightarrow \min_{\theta}$$



23. Принцип максимизации правдоподобия, его связь с эмпирическим риском.

Принцип максимизации правдоподобия предполагает, что оптимальные параметры модели можно получить путём максимизации функции правдоподобия — вероятности наблюдаемых данных при заданных параметрах модели. С точки зрения вероятностного вывода, этот подход означает выбор таких параметров модели, при которых вероятность наблюдаемой выборки максимальна.

Предположим, что есть выборка $X = \{(x_1, y_1), (x_2, y_2), \dots, (x_n, y_n)\}$, где каждый объект x – вектор параметров, y – значение целевой переменной (непрерывное значение в случае задачи регрессии и метки классов в случае задачи классификации). Пусть модель описывается параметрами θ и задаёт условное распределение вероятности отклика $p(y | x; \theta)$ (это ответ, который возвращает обучающая модель: вероятность получения отклика y при заданном входе x и параметрах θ). Функция правдоподобия (likelihood function) определяется как вероятность получить имеющуюся выборку при заданных параметрах модели:

$$L(\theta) = \prod_{i=1}^n p(y_i | x_i; \theta) \rightarrow \max_{\theta}$$

То есть мы хотим найти такие параметры θ , при которых наблюдаемые данные наилучшим образом объясняются моделью.

Чтобы избавится от произведения, можно прологарифмировать обе части и перейти к максимизации суммы логарифмов:

$$\log L(\theta) = \sum_{i=1}^n \log(p(y_i | x_i; \theta)) \rightarrow \max_{\theta}$$

Максимизация правдоподобия позволяет оценить модели, которые возвращают вероятности, в таком случае данная формулировка эквивалентна эмпирическому риску (минимизация функции потерь).

24. L1- и L2-регуляризация. Вероятностный смысл регуляризации.

См. билет 18.

25. Понятие расстояния между объектами. Метрика Минковского.

Расстояние между объектами используется для оценки степени близости объектов в пространстве признаков. Расстояние лежит в основе множества алгоритмов, таких как кластеризация, классификация (например, метод k ближайших соседей), поиск ближайших соседей, метрическое обучение и др.

Расстояние между объектами определяется на основе некоторой метрики, то есть функции, обладающей рядом формальных свойств:

1. Расстояние между любыми двумя объектами всегда больше или равно нулю;
2. Равенство нулю только при совпадении объектов $d(x_1, x_2) = 0 \iff x_1 = x_2$;
3. Симметричность: $d(x_1, x_2) = d(x_2, x_1)$;
4. Неравенство треугольника: $d(x_1, x_3) \leq d(x_1, x_2) + d(x_2, x_3)$;

- Евклидова метрика и обобщенная метрика Минковского:

$$\rho(x_i, x_k) = \left(\sum_{j=1}^n |x_{i,j} - x_{k,j}|^2 \right)^{1/2}, \quad \rho(x_i, x_k) = \left(\sum_{j=1}^n \theta_j |x_{i,j} - x_{k,j}|^p \right)^{1/p}$$

$x_i = (x_{i1}, \dots, x_{in})$ – вектор признаков объекта x_i

$x_k = (x_{k1}, \dots, x_{kn})$ – вектор признаков объекта x_k

$\theta_1, \dots, \theta_n$ – обучаемые веса (параметры) признаков, играющие две роли:

- нормировка, т.е. приведение к общему масштабу;
- задание степени важности (информативности) признаков.

Метрика Минковского показывает, насколько различны объекты i и k в признаковом пространстве, то есть измеряет степень их отличия по всем признакам, принимая во внимание как абсолютные различия между соответствующими признаками, так и степень этих различий (в зависимости от параметра p).

26. Обобщенный метрический классификатор. Метод k ближайших соседей.

Метод не подразумевает обучения в прямом смысле, в контексте данного алгоритма обучение – запоминание признаков каждого примера обучающей выборки (ленивый алгоритм или *lazy learning*). При поступлении нового объекта алгоритм находит k ближайших к нему объектов из обучающей выборки по заданной метрике расстояния (чаще всего используется евклидова метрика или метрика Минковского). Далее для задачи классификации определяется наиболее часто встречающийся класс среди этих соседей — этот класс и присваивается новому объекту.

Параметр k играет роль сглаживающего механизма: при слишком маленьком k (например, $k = 1$) алгоритм может быть чувствительным к шуму в данных, что приводит к переобучению; при слишком большом k — модель может стать слишком грубой и игнорировать локальную структуру данных, что ведёт к недообучению.

Метод k ближайших соседей (k nearest neighbours, kNN)

$w(i, x) = [i \leq 1]$ – метод ближайшего соседа

$w(i, x) = [i \leq k]$ – метод k ближайших соседей

- Преимущества:

- простота реализации (lazy learning);
- параметр k можно оптимизировать по **leave-one-out**:

$$\text{LOO}(k, X^\ell) = \sum_{i=1}^{\ell} [g(x_i; X^\ell \setminus \{x_i\}, k) \neq y_i] \rightarrow \min_k$$

- Недостатки:

- неоднозначность классификации при $\Gamma_y(x) = \Gamma_s(x), y \neq s$
- не учитываются значения расстояний



27. Метод k взвешенных ближайших соседей. Метод окна Парзена.

Классический метод k -ближайших соседей при классификации предполагает, что вклад всех соседей одинаков. Однако в реальных задачах сосед, находящийся ближе к рассматриваемому объекту, может быть более информативным, чем тот, который расположен дальше. Метод k взвешенных ближайших соседей учитывает это, назначая каждому из k соседей вес, обратно пропорциональный расстоянию до исследуемой точки.

$$w_i = \frac{1}{d(x, x_i)}$$

После этого класс объекта определяется как класс с наибольшей суммой весов среди соседей. Преимущество метода заключается в том, что он снижает влияние удалённых точек, в том числе ошибочных или шумных, и позволяет более точно учитывать локальную структуру данных.

Метод окна Парзена представляет собой метод оценки плотности вероятности с помощью функции ядра (kernel function) и используется, как правило, для построения вероятностных классификаторов. Он основан на предположении, что плотность распределения выборки можно аппроксимировать суммой сглаживающих функций (ядер), центрированных в точках обучающей выборки.

Метод окна Парзена можно рассматривать как обобщение метода k-взвешенных соседей, при котором не фиксируется число ближайших точек, а взвешиваются все объекты обучающей выборки в зависимости от расстояния до текущей точки.

Метод k взвешенных ближайших соседей

$$w(i, x) = [i \leq k]\theta_i,$$

θ_i – вес, зависящий только от номера соседа.

- Возможные эвристики:

$$\theta_i = \frac{k+1-i}{k} \text{ – линейное убывание веса;}$$

$$\theta_i = q^i \text{ – экспоненциально убывающие веса, } 0 < q < 1$$

- Проблемы:

- как более обоснованно задать веса?
- Возможно, было бы лучше, если бы вес $w(i, x)$ зависел не от порядкового номера соседа i , а от расстояния до него $\rho(x, x^{(i)})$



Метод окна Парзена

$$w(i, x) = K\left(\frac{\rho(x, x^{(i)})}{h}\right), \text{ где } h \text{ – ширина окна (bandwidth; радиус окрестности)}$$

$K(r)$ – ядро (kernel), не возрастает и положительно на $[0,1]$

- Метод парзеновского окна *фиксированной ширины*:

$$g(x; X^\ell, \mathbf{h}, K) = \arg \max_{y \in Y} \sum_{i=1}^{\ell} [y_i = y] K\left(\frac{\rho(x, x^{(i)})}{\mathbf{h}}\right)$$

- Метод парзеновского окна *переменной ширины*:

$$g(x; X^\ell, \mathbf{k}, K) = \arg \max_{y \in Y} \sum_{i=1}^{\ell} [y_i = y] K\left(\frac{\rho(x, x^{(i)})}{\rho(x, x^{(k+1)})}\right)$$

- Оптимизация параметров – по критерию LOO:

- выбор ширины окна h или числа соседей k
- выбор ядра K

28. Оптимальная разделяющая гиперплоскость, ее геометрическая интерпретация.

Пусть выборка X^ℓ линейно разделима (в пространстве признаков примеры можно разделить прямой линией), тогда существует такой вектор весов тетта и биас (смещение, на слайде тетта 0), что все метки окажутся по правильную сторону от разделяющей прямой. y_i - метка класса обучающего примера $\{-1; +1\}$, произведение фактической метки

класса и предсказание классификатора будет больше нуля, когда знаки множителей совпадают, это условие правильной классификации. Данное произведение называется отступом (зазор, margin, M). Нормировка накладывает ограничение, что минимальный отступ должен быть равен 1. Разделяющая полоса (гиперплоскость) посередине описывается следующим условием:

$$\{\vec{x}: -1 \leq \langle \vec{x}, \vec{\theta} \rangle - \theta_0 \leq 1\}$$

Это значит, что верхняя граница полосы +1, нижня -1. Точки, лежащие на границах полосы являются опорными векторами, которые определяют положение данной полосы. Ширина данной полосы описывается следующим выражением:

$$\frac{\langle \vec{x}_+ - \vec{x}_-, \vec{\theta} \rangle}{\|\vec{\theta}\|} = \frac{2}{\|\vec{\theta}\|}$$

$\|\vec{\theta}\|$ - евклидова норма или длина вектора тетта. Данный переход справедлив, так как мы наложили ограничение на отступ ≥ 1 .

Максимизация ширины данной полосы или минимизация длины вектора тетта максимизирует разделение между классами, а значит ведёт к более точному решению задачи классификации.

Линейный классификатор: $g(x; \vec{\theta}, \theta_0) = \text{sign}(\langle \vec{x}, \vec{\theta} \rangle - \theta_0)$

Пусть выборка $X^\ell = (\vec{x}_i, y_i)_{i=1}^\ell$ линейно разделима:

$$\exists \vec{\theta}, \theta_0: M_i(\vec{\theta}, \theta_0) = y_i (\langle \vec{x}_i, \vec{\theta} \rangle - \theta_0) > 0, \quad i = 1, \dots, \ell$$

Нормировка (ограничение): $\min_i M_i(\vec{\theta}, \theta_0) = 1$

Справка: $\frac{\vec{\theta}}{\|\vec{\theta}\|}$ – вектор единичной длины, с тем же направлением, что и $\vec{\theta}$

Разделяющая полоса (разделяющая гиперплоскость посередине):

$$\{\vec{x}: -1 \leq \langle \vec{x}, \vec{\theta} \rangle - \theta_0 \leq 1\}$$

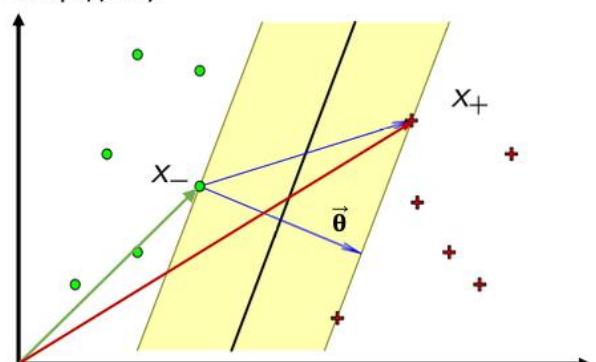
$$\exists \vec{x}_+: \langle \vec{x}_+, \vec{\theta} \rangle - \theta_0 = +1$$

$$\exists \vec{x}_-: \langle \vec{x}_-, \vec{\theta} \rangle - \theta_0 = -1$$

$\vec{\theta}$ – вектор произвольной длины, \perp разделяющей гиперплоскости

Ширина полосы:

$$\frac{\langle \vec{x}_+ - \vec{x}_-, \vec{\theta} \rangle}{\|\vec{\theta}\|} = \frac{2}{\|\vec{\theta}\|} \rightarrow \max$$



Геометрическая интерпретация

Дано:

линейно-разделимое множество объектов двух классов
 $X = X_+ \cup X_- = \mathbb{R}^n$;

$\vec{\theta}$ – вектор, перпендикулярный к разделяющей гиперплоскости;

\vec{x} – объект (вектор), класс которого неизвестен, $\vec{x} \in X$.

Найти:

класс, к которому относится \vec{x} .

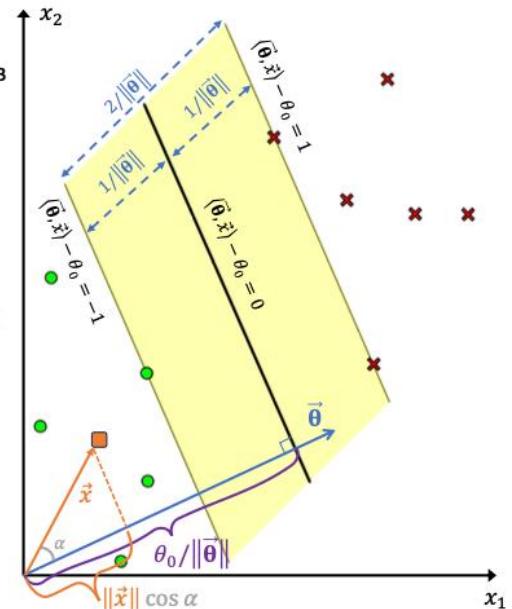
Решение:

Чтобы узнать, по какую сторону от разделяющей гиперплоскости находится объект \vec{x} , достаточно спроектировать вектор \vec{x} на вектор $\vec{\theta}$ и сравнить длину получившейся проекции с пороговым значением θ_0 :

$$\langle \vec{\theta}, \vec{x} \rangle \geq \theta_0 \Leftrightarrow \|\vec{x}\| \cos \alpha \geq \theta_0 / \|\vec{\theta}\|$$

Решающее правило:

$$\begin{aligned} \langle \vec{\theta}, \vec{x} \rangle - \theta_0 \geq 0 &\Rightarrow \vec{x} \in X_+ \\ \langle \vec{\theta}, \vec{x} \rangle - \theta_0 < 0 &\Rightarrow \vec{x} \in X_- \end{aligned}$$



29. Применение условий Каруша-Куна-Такера к задаче построения оптимальной разделяющей гиперплоскости.

Из пред. билета можно сформулировать задачу минимизации:

$$\min_{\theta, \theta_0} \frac{1}{2} \|\theta\|^2 \text{ при } M_i - 1, \forall i = 1, \dots, l$$

Квадратичная норма используется для борьбы с переобучением, делает задачу оптимизации строго выпуклой, то есть гарантирует 1 глобальный минимум в вершине параболы.

Применение условий ККТ позволяет преобразовать её к двойственной форме и получить решения с использованием только скалярных произведений.

Для осуществления перехода необходимо воспользоваться функцией Лагранжа, в общем виде она выглядит следующим образом:

$$L(x, \lambda) = f(x) + \sum_{i=1}^n \lambda_i g_i(x)$$

где $f(x)$ – функция, которую нужно оптимизировать, $g_i(x)$ – ограничения, λ_i – множитель

Лагранжа. В контексте оптимизации гиперплоскости $f(x) = \frac{1}{2} \|\theta\|^2$, а ограничения

$g(x) = M_i - 1 = y_i(\langle \vec{x}_i, \vec{\theta} \rangle - \theta_0) - 1$. Уравнение Лагранжа для оптимизации отступа гиперплоскости выглядит следующим образом:

$$L(\vec{\theta}, \theta_0, \lambda) = \frac{1}{2} \|\vec{\theta}\|^2 - \sum_{i=1}^l \lambda_i [y_i(\langle \vec{x}_i, \vec{\theta} \rangle - \theta_0) - 1]$$

где $\lambda_i \geq 0$ - множители Лагранжа для каждого примера обучающей выборки X^l .

Условия Каруша-Куна-Такера включают в себя:

1. Стационарность (градиенты по параметрам равны нулю): $\frac{\partial L}{\partial \vec{\theta}} = \vec{\theta} - \sum_{i=1}^l \lambda_i y_i \vec{x}_i = 0$,
- $$\frac{\partial L}{\partial \theta_0} = \sum_{i=1}^l \lambda_i y_i = 0;$$
2. Допустимость (выполнение исходных ограничений): $y_i(\langle \vec{x}_i, \vec{\theta} \rangle - \theta_0) - 1 \geq M_i - 1$;
3. Дополняющая нежесткость (нежесткость подразумевает, что для примеров, не являющихся опорными векторами, значение множителя Лагранжа может обратиться в 0): $\lambda_i(M_i - 1) = 0$ (если $M_i > 1$, то $\lambda_i = 0$, если $M_i = 1$, то $M_i - 1 = 0$ и $\lambda_i \neq 0$).

Из условия стационарности следует, что:

$$\vec{\theta} = \sum_{i=1}^l \lambda_i y_i \vec{x}_i \quad \langle \vec{x}, \vec{\theta} \rangle = \sum_{i=1}^l \lambda_i y_i \langle \vec{x}_i, \vec{x} \rangle$$

Из всего этого следует, что решающее правило можно сформулировать не через вектор весов θ , а через опорные вектора.

30. Понятие опорного вектора и типизация объектов.

Из предыдущего билета можно составить систему условий Каруша-Куна-Такера:

$$\begin{aligned} \frac{\partial L}{\partial \vec{\theta}} = \vec{\theta} - \sum_{i=1}^l \lambda_i y_i \vec{x}_i &= 0; \quad \frac{\partial L}{\partial \theta_0} = \sum_{i=1}^l \lambda_i y_i &= 0; \\ y_i(\langle \vec{x}_i, \vec{\theta} \rangle - \theta_0) - 1 &\geq 0; \\ \lambda_i(y_i(\langle \vec{x}_i, \vec{\theta} \rangle - \theta_0) - 1) &= 0; \\ \lambda_i &\geq 0; \end{aligned}$$

- $M_i > 1; \lambda_i = 0$ - периферийный пример (объект, который находится слишком далеко от гиперплоскости, на неё не влияет, множитель Логранжа = 0 из условия ККТ);
- $M_i = 1; \lambda_i > 0$ - опорный-граничный пример (объект, который находится на границе отступа, задаёт поведение разделяющей гиперплоскости);
- $M_i < 1; \lambda_i > 0$ - опорный-нарушитель (объект, который либо неправильно классифицирован, либо находится слишком близко к гиперплоскости).

Опорный-граничный и Опорный-нарушитель являются опорными векторами, именно они влияют на поведение гиперплоскости.

31. Нелинейное обобщение метода опорных векторов с помощью функции ядра. Виды ядер.

Из условия Каруша-Куна-Такера следует, что метка класса для объекта определяется по формуле:

$$f(x) = \operatorname{sign}\left(\sum_{i=1}^l \lambda_i y_i \langle \vec{x}_i, \vec{x} \rangle - \theta_0\right)$$

Как было сказано выше (бillet 18), данная формула была выведена из условия, что выборка X^l линейно разделима, однако, это не всегда так, и в противных случаях классификатор будет работать некорректно. Для решения этой проблемы существует Ядерный трюк (kernel-trick).

Функция ядра — это функция, которая вычисляет скалярное произведение в некотором (возможно бесконечномерном) пространстве признаков без необходимости явного перехода к этому пространству. Применяя ядерную функцию вместо обычного скалярного произведения признаков, осуществляется переход к спрямляющему пространству, обычно более высокой размерности, в котором уже, возможно, данные могут стать линейно разделимыми.

Идея: заменить $\langle x, x' \rangle$ нелинейной функцией $K(x, x')$.

Переход к спрямляющему пространству, как правило более высокой размерности: $\psi: X \rightarrow H$, т.е. ψ – функция для преобразования из X в H .

Определение:

Функция $K: X \times X \rightarrow \mathbb{R}$ – ядро, если $K(x, x') = \langle \psi(\vec{x}), \psi(\vec{x}') \rangle$ при некотором $\psi: X \rightarrow H$, где H – гильбертово пространство.

Теорема:

Функция $K(x, x')$ является ядром тогда и только тогда, когда она

- симметрична: $K(x, x') = K(x', x)$;
- и неотрицательно определена: $\int_X \int_X K(x, x') g(x)g(x') dx dx' \geq 0$ для любой $g: X \rightarrow \mathbb{R}$.

Примеры ядер

1. Линейное ядро

$$K(x, x') = \langle x, x' \rangle$$

2. Квадратичное ядро

$$K(x, x') = \langle x, x' \rangle^2$$

3. Полиномиальное ядро с произведениями (одночленами) степени d

$$K(x, x') = \langle x, x' \rangle^d$$

4. Полиномиальное ядро с одночленами степени $\leq d$

$$K(x, x') = (\langle x, x' \rangle + 1)^d$$

5. Нейросеть с сигмоидными функциями активации

$$K(x, x') = \text{th}(k_1 \langle x, x' \rangle - k_0), \quad k_0, k_1 \geq 0$$

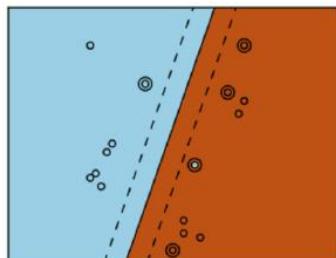
6. Сеть радиальных базисных функций (RBF ядро, гауссовское ядро)

$$K(x, x') = \exp(-\gamma \|x - x'\|^2)$$

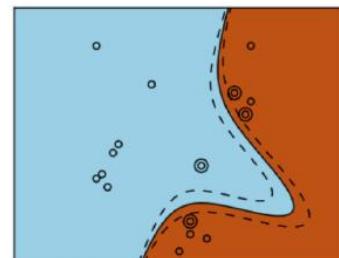
Гиперплоскость в спрямляющем пространстве соответствует нелинейной разделяющей поверхности в исходном.

Примеры с различными ядрами $K(x, x')$

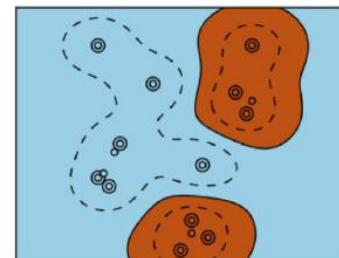
линейное
 $\langle x, x' \rangle$



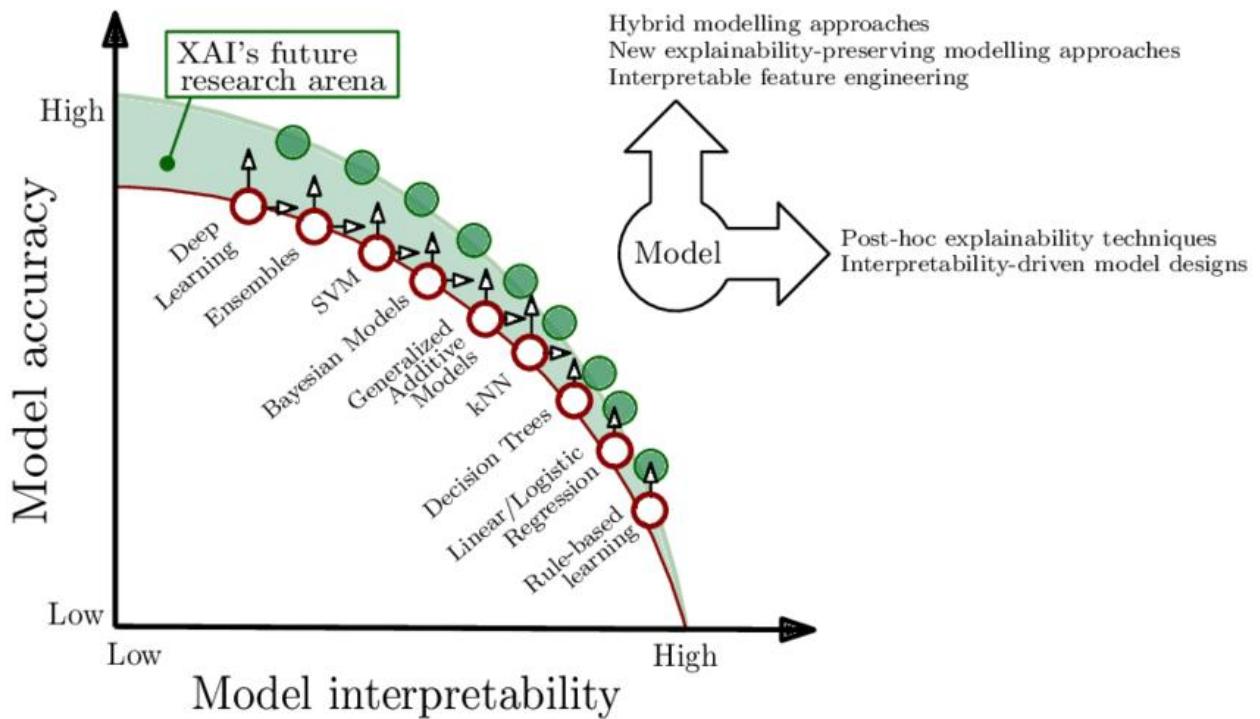
полиномиальное
 $(\langle x, x' \rangle + 1)^d, \quad d=3$



гауссовское (RBF)
 $\exp(-\gamma \|x - x'\|^2)$



32. Интерпретируемость алгоритмов машинного обучения.



Интерпретируемость алгоритмов машинного обучения — это свойство модели, позволяющее человеку понять, каким образом она принимает решения.

XAI, eXplainable AI обозначает совокупность методов, подходов и инструментов, позволяющих сделать поведение систем искусственного интеллекта более прозрачным и понятным для человека. Основная цель XAI заключается в обеспечении возможности для пользователя анализировать, интерпретировать и верифицировать результаты, полученные интеллектуальной системой, особенно в случаях использования моделей так называемого чёрного ящика.

- Interpretability – Способность человека понять внутреннее устройство модели или причину её конкретного предсказания. Это означает, что можно проследить, как модель пришла к выводу: какие признаки использовались, как они повлияли на результат и почему.
- Understandability, Transparency – Простота конструкции модели и возможность полностью изучить её поведение без необходимости дополнительных инструментов. Прозрачная модель – это модель, чью структуру можно просмотреть, логически разобрать и понять. Transparency означает доступность всех внутренних механизмов модели (например, весов, формул), а understandability – то, насколько они понятны человеку.

- Explainability – Способность модели или внешнего метода предоставить человеку объяснение, понятное и обоснованное, относительно того, как было получено предсказание.
- Comprehensibility – Способность не просто прочитать и объяснить результат, а глубоко осознать всю картину принятия решений моделью, включая её поведение в разных случаях, логику предсказаний и причинно-следственные связи. Это наиболее высокий уровень понимания, предполагающий, что человек не только получил объяснение, но и может с уверенностью использовать и интерпретировать модель в контексте реальной задачи.

33. Деревья принятия решений. Определение, алгоритмы построения.

Дерево принятия решений (Decision Tree) – один из наиболее интерпретируемых и понятных моделей классификации. Основная идея дерева заключается в последовательном разбиении признакового пространства на подпространства, в каждом из которых принимается локальное решение. На каждом узле дерева выполняется проверка условия на значение одного из признаков, и в зависимости от результата объект направляется либо в левую, либо в правую ветвь дерева.

Корень (root) — это начальный узел, внутренние узлы (internal nodes) — точки, в которых принимаются решения о разбиении, листья (leaves) — конечные узлы, соответствующие предсказаниям классов или значений.

В каждой вершине дерева признак проверяется на условие $f_i(x) \leq a$, и на основе результата этого условия переходим на нижний уровень дерева, пока не дойдём до листа дерева.

Определение решающего дерева (Decision Tree)

Решающее дерево – алгоритм классификации $g(x)$, задающийся деревом (связным ациклическим графом) с корнем $v_0 \in V$ и множеством вершин $V = V_{\text{внутр}} \sqcup V_{\text{лист}}$;

$f_v: X \rightarrow D_v$ – дискретный признак, $\forall v \in V_{\text{внутр}}$;

$S_v: D_v \rightarrow V$ – множество дочерних вершин;

$y_v \in Y$ – метка класса, $\forall v \in V_{\text{лист}}$;

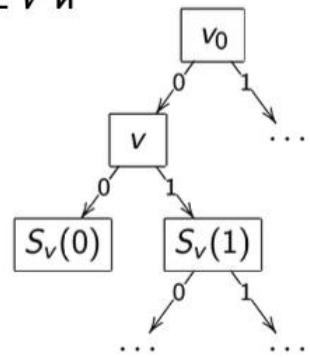
$v := v_0$;

пока $(v \in V_{\text{внутр}})$: $v := S_v(f_v(x))$;

вернуть $g(x) := y_v$;

Чаще всего используются бинарные признаки вида $f_v(x) = [f_j(x) \geq a]$

Если $D_v \equiv \{0,1\}$, то решающее дерево называется **бинарным**



CART (Classification and Regression Trees) — наиболее широко используемый алгоритм, позволяющий строить как классификационные, так и регрессионные деревья. В классификации использует критерий Джини или энтропийный критерий (см. билет 34), в регрессии — минимизацию среднеквадратичной ошибки (MSE). CART всегда строит бинарные деревья.

ID3 (Iterative Dichotomiser 3) — основан на выборе признака с максимальной информационной выгодой (information gain), измеряемой на основе энтропии. Прирост информации - разница между энтропией до и после разбиения (см. билет 34).

Преимущества деревьев принятия решений включают интерпретируемость, способность работать с данными разного типа, нечувствительность к масштабированию признаков и небольшим изменениям в данных. Недостатками являются склонность к переобучению и нестабильность (небольшие изменения в данных могут привести к совершенно иному дереву).

34. Критерий Джинни, энтропийный критерий.

Данные критерии оценивают степень однородности классов и объектов в подмножестве, чем выше однородность, тем лучше будет разбиение.

Критерий Джини измеряет вероятность ошибочной классификации случайного элемента, если классифицировать его по распределению классов внутри узла. Для набора объектов с классами c_1, c_2, \dots, c_k , критерий Джини определяется как:

$$G = 1 - \sum_{i=1}^k p^2(c_i)$$

То есть, если узел покрывает только один класс, то вероятность неправильно классифицировать этот класс нулевая, критерий Джини = 0, если узел покрывает два класса, критерий Джини = 0,5 и тп. Дерево стремится разбивать выборку так, чтобы значения Джини в дочерних узлах были как можно ниже — это означает, что классы внутри этих узлов распределены более однородно.

Энтропийный критерий основан на понятии энтропии из теории информации. Он измеряет количество неопределенности в распределении классов. (Для справки: данный критерий опирается на энтропию Шеннона: $H(X_m) = -\sum p_i \log p_i$. Так как вероятность $p_i \in [0,1]$, энтропия не отрицательна. Если случайная величина принимает только одно значение, то величина абсолютно предсказуема и её энтропия равна -1, наибольшее значение энтропия принимает для равно распределённой случайной величины, это значит, что значение наиболее сложно предсказать).

Энтропия узла рассчитывается по формуле:

$$\text{Entropy} = -\sum p_i \log_2 p_i$$

Логарифм по основанию 2 отражает минимальное число двоичных (да/нет) вопросов, необходимых для определения класса объекта. Например, если энтропия равна 1 биту, это означает, что в среднем один двоичный вопрос (или проверка) необходим, чтобы однозначно классифицировать объект.

Из энтропии можно получить метрику Прирост информации (Information Gain), Данная метрика показывает, насколько сильно уменьшается неопределенность (энтропия) целевой переменной при разделении данных по конкретному признаку. То есть, насколько хорошо признак помогает «приблизиться» к правильной классификации.

$$\text{InformationGain} = \text{Entropy}(\text{parent}) - \frac{k_i}{k} \text{Entropy}(\text{child}_i)$$

где k_i - количество объектов в i -ом подмножестве, k – общее количество объектов.

35. Проблема переобучения деревьев принятия решений. Регулирование глубины дерева (обрезка ветвей).

Переобучение (overfitting) деревьев принятия решений представляет собой ситуацию, при которой построенная модель слишком точно отражает особенности обучающей выборки, включая шум и случайные выбросы, в ущерб обобщающей способности. Такое дерево, как правило, имеет большую глубину, множество ветвей и разделяет данные до уровня полной чистоты узлов, что не требуется, для сохранения обобщающей способности.

Для борьбы с переобучением применяется регулирование сложности дерева, одним из основных методов которого является обрезка дерева (pruning). Обрезка направлена на удаление ненужных или малозначимых ветвей, которые не дают ощутимого вклада в точность модели на тестовых данных.

Предварительная обрезка (pre-pruning). Она включает в себя установку ограничений на процесс построения дерева, таких как: максимальная глубина дерева, минимальное количество объектов в узле для дальнейшего разбиения, минимальное уменьшение критерия чистоты (значение критерия Джини или Энтропийного критерия) для выполнения разбиения. Эти параметры ограничивают рост дерева на этапе построения, предотвращая появление переобобщающих ветвей. Однако при слишком агрессивной предварительной обрезке возможна потеря информации, что приводит к недообучению.

Построенная обрезка (post-pruning). Она применяется после построения полного дерева и заключается в удалении или упрощении уже существующих ветвей. Основные методы post-pruning: сокращение ветвей, если они не улучшают метрику качества на тестовой выборке, обрезка на основе оценки статистической значимости разбиений и критериев чистоты. Построенная обрезка считается более точным методом, поскольку она основана на оценке уже построенной структуры, однако требует дополнительного этапа валидации и усложняет вычисления.

36. Объясните понятие ошибок первого и второго рода, их связь с машинным обучением.

Рассмотрим понятие ошибки первого и второго рода в контексте задачи бинарной классификации. Существует два класса, Положительный (True) и Отрицательный (False) классы. Также существует две гипотезы, принадлежность отрицательному и положительному классу, соответственно гипотезы 0 и 1.

Ошибка первого рода - это ложноположительное решение. Она возникает в ситуации, когда модель отвергает нулевую гипотезу, хотя на самом деле она истинна. В контексте бинарной классификации это означает, что модель предсказала положительный результат, хотя истинная метка отрицательная. В классификационных задачах эта ошибка связана с метрикой False Positive (FP) — количество отрицательных примеров, ошибочно классифицированных как положительные.

Ошибка второго рода — это ложноотрицательное решение. Она возникает, когда модель не отвергает нулевую гипотезу, хотя она ложна. В терминах бинарной классификации это означает, что объект ошибочно отнесен к отрицательному классу, в то время как он должен быть классифицирован как положительный. Соответствует метрике False Negative (FN) — количество положительных объектов, ошибочно классифицированных как отрицательные.

В машинном обучении, на основе ошибок первого и второго рода, формируется матрица ошибок (confusion matrix), которая отражает соотношения между предсказанными и истинными метками классов:

		PREDICTED	
		Positive	Negative
ACTUAL	Positive	True Positive (TP)	False Negative (FN)
	Negative	False Positive (FP)	True Negative (TN)

37. Объясните понятия accuracy, полноты (recall), точности (precision) и F1-меры.

Accuracy (доля верных классификаций) — это обобщённая метрика, определяемая как доля правильно классифицированных объектов среди всех наблюдений:

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

Метрика показывает общее качество классификатора, однако она неустойчива к дисбалансу классов, поскольку высокая точность может достигаться при игнорировании редкого, но важного класса. **Accuracy не является точностью классификации, не смотря на название.**

Precision (точность) — это доля правильно классифицированных положительных объектов среди всех объектов, предсказанных как положительные:

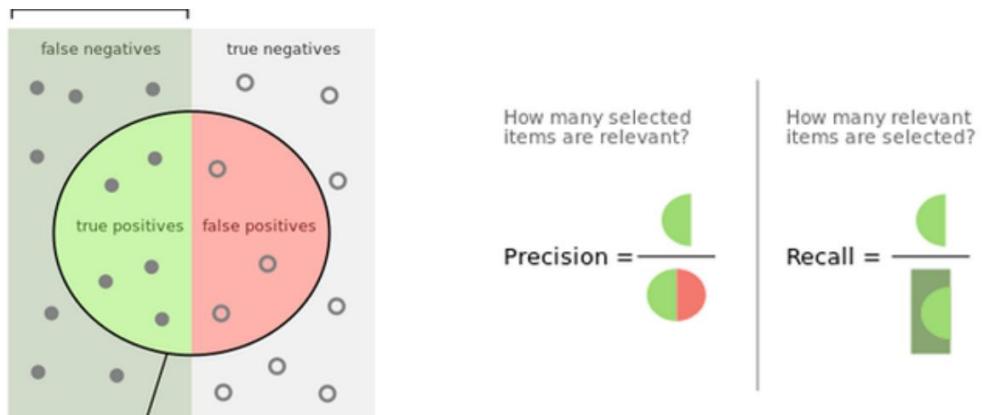
$$precision = \frac{TP}{TP + FP}$$

Метрика показывает, насколько модель склонна к ложноположительным ошибкам. Высокая точность означает, что среди объектов, отнесённых моделью к положительному классу, преобладают действительно положительные примеры.

Recall (полнота, чувствительность) — это доля правильно классифицированных положительных объектов среди всех реально положительных объектов:

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

Метрика показывает, насколько хорошо модель обнаруживает объекты положительного класса. Высокая полнота означает, что модель способна обнаруживать большинство положительных случаев, даже если при этом возникают ложноположительные ошибки.



F1-мера — это среднее гармоническое между точностью и полнотой, обеспечивающее компромисс между этими двумя метриками:

$$F1 = 2 * \frac{precision * recall}{precision + recall}$$

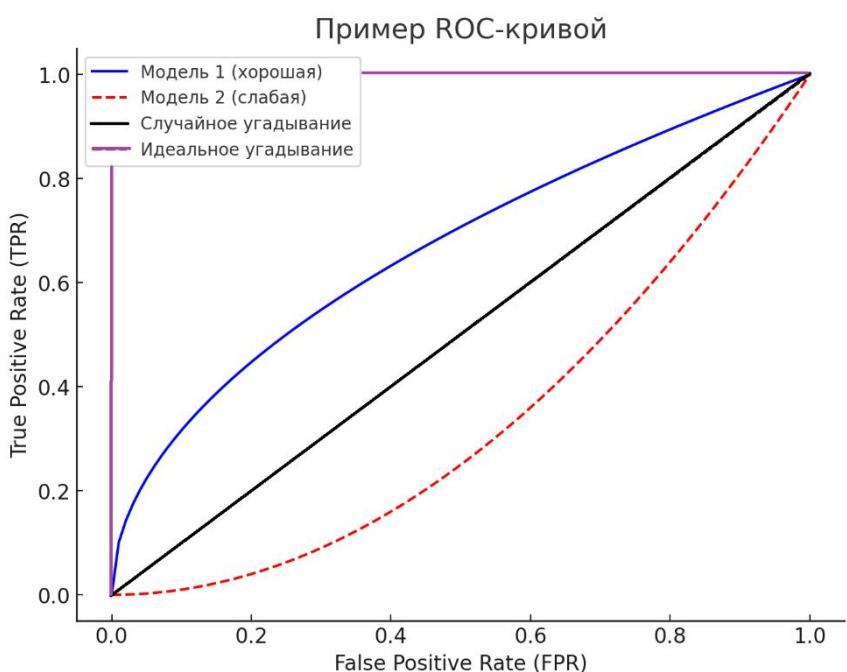
Данная метрика особенно полезна при дисбалансе классов, когда важно учитывать одновременно и полноту, и точность. Низкое значение одной из составляющих существенно снижает итоговое значение F1-метрики.

38. Кривые ROC и Precision-Recall, площадь под ними.

Кривая ROC – это график, который иллюстрирует производительность классификационной модели при всех возможных порогах классификации. Ось X данного графика представляет собой FPR, т.е ложноположительную частоту, а ось Y — TPR, т.е истинноположительную частоту. TPR также известен как Recall, и определяется как доля правильно классифицированных положительных результатов относительно всех положительных результатов в данных. FPR определяет долю ошибочно классифицированных отрицательных результатов относительно всех отрицательных результатов и вычисляется как:

$$FPR = \frac{FP}{TN + FP}$$

Кривая ROC представляет собой графическое представление компромисса между чувствительностью и специфичностью при различных порогах классификации. Идеальная модель классификации будет стремиться к точке в верхнем левом углу графика, где TPR равно 1, а FPR равно 0.

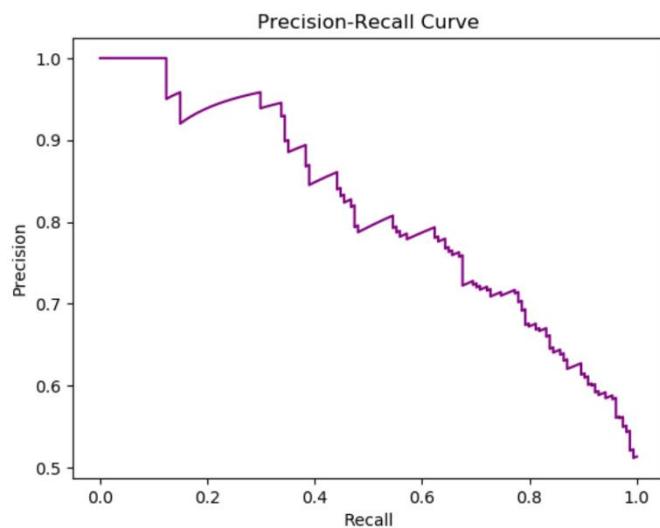


Показатель AUC (Area Under the ROC Curve) — это мера, которая позволяет суммировать производительность модели одним числом, измеряя площадь под кривой

ROC. AUC колеблется от 0 до 1, где более высокое значение AUC указывает на более высокую производительность модели.

Precision-Recall кривая строится как график зависимости точности (Precision) от полноты (Recall) при изменении порога. Эта кривая особенно полезна при работе с несбалансированными данными, когда положительный класс значительно меньше отрицательного. В таких случаях ROC-кривая может быть оптимистичной, так как FPR может оставаться низкой за счёт большого числа отрицательных примеров, тогда как Precision-Recall лучше отражает качество работы с редкими классами.

Площадь под Precision-Recall кривой (Average Precision, AP) также служит метрикой качества классификатора в задачах с дисбалансом классов. Высокое значение AP указывает на то, что модель способна одновременно обеспечивать высокую полноту при сохранении хорошей точности.



39. Метрики оценки качества регрессии.

- MSE (Mean Squared Error): Она рассчитывается как среднее арифметическое квадратов разностей между предсказанными и истинными значениями. MSE подчёркивает большие ошибки за счёт квадратичного возведения, что делает её чувствительной к выбросам. $MSE = \frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}_i)^2$
- MAE (Mean Absolute Error): это среднее значение абсолютных разностей между предсказанными и фактическими значениями. В отличие от MSE, MAE равномерно

учитывает все ошибки, без усиления влияния крупных отклонений, что делает её

$$\text{более устойчивой к выбросам. } MAE = \frac{1}{n} \sum_{i=1}^n |y_i - \hat{f}_i|$$

- RMSE (Root Mean Squared Error): это квадратный корень из MSE. Она возвращает ошибку в тех же единицах, что и исходные данные, что облегчает интерпретацию. RMSE также усиливает влияние больших ошибок, как и MSE.

$$RMSE = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{f}_i)^2}$$

- R-squared (R^2) Score: характеризует долю дисперсии зависимой переменной, объясненную моделью. Значения R^2 варьируются от минус бесконечности до 1, где 1 означает идеальное совпадение модели с данными, а значения близкие к нулю или отрицательные указывают на плохую предсказательную способность модели. R^2 является безразмерной метрикой и часто используется для сравнения

$$\text{моделей. } R^2 = 1 - \frac{\sum_{i=1}^n (y_i - \hat{f}_i)^2}{\sum_{i=1}^n (y_i - \bar{y})^2} . \text{ В числителе: сумма квадратов разностей между}$$

истинными значениями y_i и предсказанными значениями \hat{y} , в знаменателе: сумма квадратов разностей между истинными значениями y_i и их средним значением \bar{y} .

40. Задача кластеризации. Типы кластерных структур, чувствительность к выбору признаков.

Постановка задачи кластеризации

Дано:

- X – пространство объектов;
- $X^\ell = \{x_1, \dots, x_\ell\}$ – обучающая выборка;
- $\rho: X \times X \rightarrow [0, \infty)$ – функция расстояния между объектами

Найти:

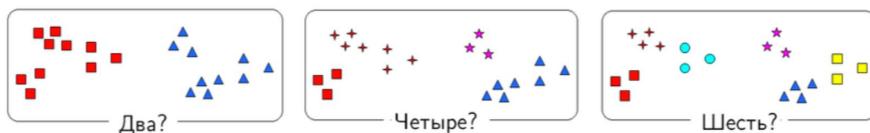
- Y – множество кластеров,
- $g: X \rightarrow Y$ – алгоритм кластеризации, такой что:
 - каждый кластер состоит из близких объектов;
 - Объекты разных кластеров существенно различны.

Это задача обучения без учителя (unsupervised learning).

Решение задачи кластеризации принципиально неоднозначно:

- точной постановки задачи кластеризации нет;
- существует много критериев качества кластеризации;
- существует много эвристических методов кластеризации;
- число кластеров $|Y|$, как правило, неизвестно заранее;
- результат кластеризации сильно зависит от метрики ρ , выбор которой также является эвристикой.

Пример: сколько здесь кластеров?



- Упростить дальнейшую обработку данных, разбить множество X^ℓ на группы схожих объектов чтобы работать с каждой группой в отдельности (задачи классификации, регрессии, прогнозирования).
- Сократить объём хранимых данных, оставив по одному представителю от каждого кластера (задачи сжатия данных).
- Выделить нетипичные объекты, которые не подходят ни к одному из кластеров (задачи одноклассовой классификации).
- Построить иерархию множества объектов, пример — классификация животных и растений К.Линнея (задачи таксономии).

41. Задача частичного обучения.

Задача частичного обучения возникает в том случае, когда выборка содержит как размеченные, так и неразмеченные данные. Цель — построить обобщающую модель, способную классифицировать новые, ранее не встречавшиеся объекты. Задача частичного обучения стоит между обучением с учителем и без учителя.

Дано:

множество объектов X , множество классов Y ;

$X^k = \{x_1, \dots, x_k\}$ – размеченные объекты (labeled data);

$\{y_1, \dots, y_k\}$

$U = \{x_{k+1}, \dots, x_\ell\}$ – неразмеченные объекты (unlabeled data);

Два варианта постановки задачи:

- Частичное обучение (semi-supervised learning, SSL):
построить алгоритм классификации $g: X \rightarrow Y$
- Трансдуктивное обучение (transductive learning):
зная **все** $\{x_{k+1}, \dots, x_\ell\}$, получить метки $\{a_{k+1}, \dots, a_\ell\}$.

Типичные приложения:

Классификация и каталогизация текстов, изображений и т.п.

42. Оценка качества решения задачи кластеризации.

Пусть известны только попарные расстояния между объектами, $a_i = a(x_i)$ - кластеризация объекта x_i .

Среднее внутрикластерное расстояние:

$$F_0 = \frac{\sum_{i=1}^n \sum_{j=i+1}^n d(x_i; x_j) / (a_i = a_j)}{\binom{n}{2}} \rightarrow \min$$

Сумма расстояний между точками из одного и того же кластера делится на количество пар точек, принадлежащих к одному кластеру. Решая задачу кластеризации, мы хотим по возможности получать как можно более кучные кластеры, то есть минимизировать F_0 .

Среднее межкластерное расстояние:

$$F_1 = \frac{\sum_{i=1}^n \sum_{j=i+1}^n d(x_i; x_j) / (a_i \neq a_j)}{\binom{n}{2}} \rightarrow \max$$

Среднее межкластерное расстояние, напротив, нужно максимизировать, то есть имеет смысл выделять в разные кластеры наиболее удалённые друг от друга объекты.

Задача кластеризации в векторном пространстве представляет собой процесс автоматического разделения множества объектов (точек данных), представленных в виде векторов признаков.

Сумма средних внутрикластерных расстояний в линейном векторном пространстве:

$$F_0 = \frac{1}{a \in Y} \sum_{|X_a|} d(x_i, \mu_a) \rightarrow \min$$

где Y – множество кластеров, $|X_a|$ - кластер a , μ_a - центр масс кластера

$$a: \mu_a = \frac{1}{n} \sum_{i=1}^n \vec{x}_i$$

Сумма средних межкластерных расстояний в линейном векторном пространстве:

$$F_1 = \max_{a,b \in Y} d(\mu_a, \mu_b) \rightarrow \max$$

Коэффициент силуэта:

$$S(i) = \frac{R_i - r_i}{\max(R_i, r_i)} \rightarrow \max$$

где R_i - минимальное среднее расстояние до чужого кластера, r_i - среднее расстояние до объектов своего класса. Коэффициент принимает значения от -1 до $+1$ и максимизируется, когда кластеры кучные и хорошо отделены друг от друга.

BCubed-меры применяются при наличии истинной разметки, чтобы оценить соответствие кластеров истинным классам. Они определяются для каждого объекта путём сравнения разметки и кластеризации.

Точность и полнота кластеризации в сравнении с эталоном

$y_i \in Y_0$ — эталонная классификация объектов, $i = 1, \dots, \ell$
 Y_0 может не совпадать с Y по мощности

$P_i = \{k: a_k = a_i\}$ — кластер объекта x_i ;
 $Q_i = \{k: y_k = y_i\}$ — эталонный класс объекта x_i

BCubed-меры точности и полноты кластеризации:

$$\text{Precision} = \frac{1}{\ell} \sum_{i=1}^{\ell} \frac{|P_i \cap Q_i|}{|P_i|} \text{ — средняя точность}$$

$$\text{Recall} = \frac{1}{\ell} \sum_{i=1}^{\ell} \frac{|P_i \cap Q_i|}{|Q_i|} \text{ — средняя полнота}$$

$$F_1 = \frac{1}{\ell} \sum_{i=1}^{\ell} \frac{2|P_i \cap Q_i|}{|P_i| + |Q_i|} \text{ — средняя } F_1\text{-мера}$$

43. Метод k-средних.

Цель метода — минимизировать внутрикластерную дисперсию, то есть сумму квадратов расстояний между объектами и центрами соответствующих кластеров. Алгоритм относится к методам жёсткой кластеризации: каждый объект однозначно относится к одному кластеру.

Минимизация суммы квадратов внутрикластерных расстояний:

$$\sum_{i=1}^{\ell} \|x_i - \mu_{a_i}\|^2 \rightarrow \min_{\{a_i\}, \{\mu_a\}}, \quad \|x_i - \mu_a\|^2 = \sum_{j=1}^n (f_j(x_i) - \mu_{aj})^2$$

Алгоритм метода k-средних (алгоритм Лойда):

1. Случайным образом выбираются k начальных центров кластеров μ_1, \dots, μ_k .
2. Каждый объект x_i назначается тому кластеру, чей центр ближе всего.
3. Центры кластеров пересчитываются как средние значения объектов, вошедших в кластер.
4. Повторение шагов 2–3 до сходимости: алгоритм считается сущедшимся, если центры масс не изменяются или изменения становятся меньше заданного порога.

Алгоритм гарантированно сходится за конечное число шагов, так как функция потерь (сумма квадратов расстояний) не увеличивается при каждом обновлении. Однако он может сходиться к локальному минимуму, что делает важным выбор начальных центров.

Изменения в алгоритме Лойда для частичного обучения:

1. Если известны метки части объектов, можно использовать их для инициализации соответствующих центров кластеров: Вычислить центры кластеров по размеченным данным.
2. Размеченные объекты не переклассифицируются — их принадлежность к кластерам фиксирована в соответствии с их метками, а неразмеченные объекты назначаются в кластеры на основе минимального расстояния до центров.
3. Центры пересчитываются по всем объектам, иногда добавляется вес для уже размеченных объектов, чтобы сделать их более значимыми.
4. Повторяются шаги 2 и 3 до сходимости.

44. Алгоритм DBSCAN.

Алгоритм DBSCAN (Density-Based Spatial Clustering of Applications with Noise) представляет собой метод кластеризации, основанный на плотности объектов в пространстве признаков. Он выявляет кластеры как области высокой плотности, разделённые областями низкой плотности, что позволяет эффективно обнаруживать кластеры произвольной формы, а также отделять шумовые или аномальные точки.

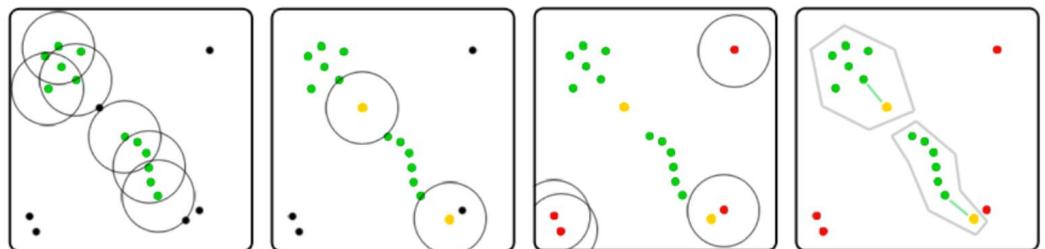
Принцип работы DBSCAN основывается на двух гиперпараметрах: ε (эпсилон) — радиус окрестности, и m — минимальное число точек, необходимых для образования кластера.

Density-Based Spatial Clustering of Applications with Noise

Объект $x \in U$, его ε -окрестность $U_\varepsilon(x) = \{u \in U: \rho(x, u) \leq \varepsilon\}$

Каждый объект может быть одного из трёх типов:

- корневой: имеющий плотную окрестность, $|U_\varepsilon(x)| \geq m$
- граничный: не корневой, но в окрестности корневого
- шумовой (выброс): не корневой и не граничный



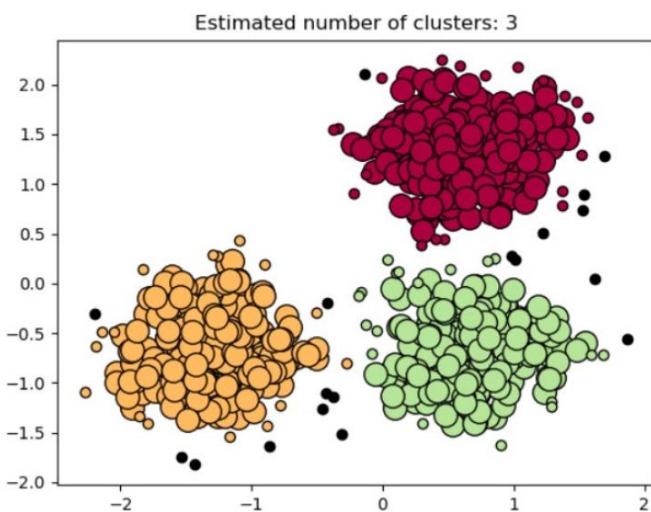
Алгоритм кластеризации DBSCAN:

1. Все точки помечаются как непомеченные, счетчик кластеров $a = 0$.
2. Пока есть непомеченные точки: берется случайная непомеченная точка x , вычисляется количество точек в её ε -окрестности $|U_\varepsilon(x)|$.
3. Если $|U_\varepsilon(x)| < m$, то точка x помечается как шумовая, иначе создается новый кластер K .
4. К инициализируется ε -окрестностью точки x : $K := U_\varepsilon(x)$, увеличивается счетчик кластеров: $a := a + 1$, цикл для каждой непомеченной или шумовой точки x' в кластере K : если $|U_\varepsilon(x')| \geq m$, то K расширяется: $K := K \cup U_\varepsilon(x')$, иначе x' помечается как граничная точка кластера K .
5. Все точки $x_i \in K$ получают метку кластера $a_i := a$.

Кластеры в DBSCAN представляют собой соединённые области с плотностью выше заданного порога. Алгоритм не требует указания количества кластеров, в отличие от k-means. Он чувствителен к выбору параметров, особенно радиуса ϵ : слишком малое значение может привести к избыточному числу шумов, слишком большое — к слиянию различных кластеров.

Преимущества алгоритма:

- быстрая кластеризация больших данных:
 $O(\ell^2)$ в худшем случае,
 $O(\ell \ln \ell)$ при эффективной реализации $U_\epsilon(x)$;
- кластеры произвольной формы (долой центры!);
- деление объектов на корневые, граничные, шумовые.



45. Иерархическая кластеризация.

Алгоритм иерархической кластеризации Ланса-Уильямса (1967 г.):

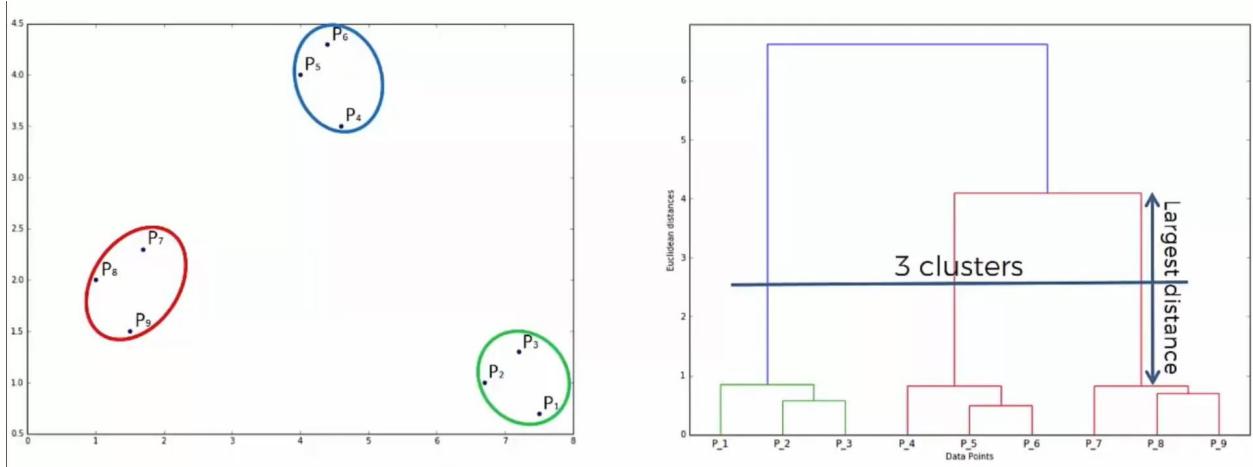
1. Начинаем с C_1 - множества одноэлементных кластеров: каждая точка данных образует отдельный кластер. Вычисляем расстояния $R\{x_i\} \{x_j\} = \rho(x_i, x_j)$ между всеми парами точек.
2. Цикл оп всем точкам, начиная со второй:
 - Поиск ближайших кластеров: В текущем множестве кластеров C_{t-1} находится пара кластеров (U, V) с минимальным расстоянием R_{UV} (в задаче

частичного обучения добавляется условие, что в U и V не должно быть объектов с разными метками);

- Создается новый кластер $W := U \cup V$ (который является объединением двух кластеров U и V);
- Обновляется множество кластеров: $C_t := C_{t-1} \cup \{W\} \setminus \{U, V\}$ (Всего кластеров становится меньше на W);
- Для каждого кластера $S \in C_t$ вычисляется расстояние до нового кластера W по формуле Ланса-Уильямса: $R_{ws} := \alpha_U R_{us} + \alpha_V R_{vs} + \beta_V R_{uv} + \gamma |R_{us} - R_{vs}|$ (данная формула позволяет вычислить расстояние от нового объединенного кластера W до любого другого кластера S через параметры: α_U , α_V - весовые коэффициенты для кластеров U и V ; β - коэффициент, учитывающий расстояние между объединяемыми кластерами; γ - коэффициент, учитывающий абсолютную разность расстояний между другим кластером S и объединёнными кластерами U и V . Изменяя параметры α , β , γ , получаем различные методы: Одиночная связь (single linkage): $\alpha_U = \alpha_V = 0.5$, $\beta = 0$, $\gamma = -0.5$; Полная связь (complete linkage): $\alpha_U = \alpha_V = 0.5$, $\beta = 0$, $\gamma = 0.5$; Средняя связь (average linkage): $\alpha_U = |U|/(|U|+|V|)$, $\alpha_V = |V|/(|U|+|V|)$, $\beta = 0$, $\gamma = 0$).

3. Алгоритм продолжается до тех пор, пока не останется один кластер, содержащий все точки. Результатом является дендрограмма - древовидная структура, показывающая порядок объединения кластеров.

Если обобщить алгоритм, то на каждой итерации мы находим минимальное расстояние между кластерами и объединяем два самых близайших кластера в один большой кластер, делаем это до того момента, пока не останется один большой кластер, содержащий все точки выборки.



Дендрограмма визуализирует насколько далеко находились объединённые между собой кластеры, чем выше разница между узлами дендрограммы, тем больше расстояние между кластерами, а значит они более разделимы. После получения дендрограммы задача выбрать такое разделение кластеров, после которого кластеры имеют максимальное расстояние.

Основные свойства иерархической кластеризации

- *Монотонность*: дендрограмма не имеет самопересечений, при каждом слиянии расстояние между объединяемыми кластерами только увеличивается: $R_2 \leq R_3 \leq \dots \leq R_\ell$.
- *Сжимающее расстояние*: $R_t \leq \rho(\mu_U, \mu_V)$, $\forall t$.
- *Растягивающее расстояние*: $R_t \geq \rho(\mu_U, \mu_V)$, $\forall t$

Теорема (Миллиган, 1979)

Кластеризация монотонна, если выполняются условия

$$\alpha_U \geq 0, \quad \alpha_V \geq 0, \quad \alpha_U + \alpha_V + \beta \geq 1, \quad \min\{\alpha_U, \alpha_V\} + \gamma \geq 0.$$

R^U не монотонно; R^6 , R^d , R^r , R^y — монотонны.

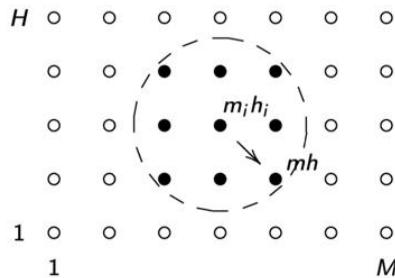
R^6 — сжимающее; R^d , R^y — растягивающие;

46. Карты Кохонена.

Карты Кохонена (Self-Organizing Maps, SOM) — это тип искусственной нейронной сети для обучения без учителя, которая создает упорядоченное представление многомерных данных на низкоразмерной сетке.

Изначально задаётся прямоугольная сеть нейронов-кластеров вида $Y = \{1, \dots, M\} \times \{1, \dots, H\}$, где M и H – количество нейронов в сетке. Каждому узлу (m, h) задаётся вектор весов $\theta_{mh} \in \mathbb{R}^n$, размерность весов соответствует размерности входных данных. На сетке определяется евклидово расстояние между узлами: $r((m_i, h_i), (m, h)) = \sqrt{(m - m_i)^2 + (h - h_i)^2}$ – расстояние между i -ым нейроном и любым другим в сетке Кохонена.

Окрестность (m_i, h_i) :



Алгоритм построения карты Кохонена:

1. $\theta_{mh} := \text{random} \in \left[-\frac{1}{2MH}, \frac{1}{2MH} \right]$ – инициализация весов для каждого нейрона.
2. Для каждого входного вектора x_i вычисляется расстояние от него до всех нейронов в сетке, выбирается нейрон с минимальным расстоянием и этот нейрон становится нейроном-победителем: $(m_i, h_i) := g(x_i) = \arg \min r(x_i, \theta_{mh})$ где r — функция расстояния между входом и весами нейрона, $\arg \min$ — "аргумент минимума" (координаты, при которых достигается минимум), $(m, h) \in Y$ — перебираем все нейроны в сетке, (m_i, h_i) — координаты нейрона-победителя. (Winner Take All или WTA).
3. Обновляются веса не только победителя, но и его соседей: для всех $(m, h) \in \text{Окрестность } (m_i, h_i)$: $\theta_{mh} := \theta_{mh} + \eta(x_i - \theta_{mh})K(r(m_i, h_i), (m, h))$, где θ_{mh} — веса нейрона с координатами (m, h) , η — темп обучения (learning rate), $(x_i - \theta_{mh})$ — вектор ошибки, $K(\dots)$ — функция соседства (функция, которая определяет силу обновления весов в зависимости от значения расстояния, обычно используется гауссова функция, которая со временем сужается, что позволяет сначала формировать общую структуру, а затем уточнять детали), $r(\dots)$ — расстояние от нейрона до победителя.
4. Пункты 2 и 3 повторяются, пока кластеризация не стабилизируется.

Сеть автоматически организует нейроны так, что похожие входные данные активируют близко расположенные нейроны. Это создает топологическое сохранение — близкие в исходном пространстве данные остаются близкими на карте.

Существует 2 способа визуализации карт Кохонена:

1. Карта плотности: в данном случае цвет узла (m, h) показывает как часто нейрон становился победителем. Темные области — много данных попадает в эти нейроны, Светлые области — мало данных. Данный способ визуализации показывает основные кластеры в данных.
2. Компонентные карты: Цвет узла (m, h) = значение j -й компоненты вектора весов θ_{mh} . Каждая карта показывает, как один конкретный признак распределен по сетке, разные цвета - разные значения признака.

Достоинства:

- Возможность визуального анализа многомерных данных
- Квантование выборки по кластерам, с автоматическим определением числа непустых кластеров

Недостатки:

- **Субъективность.** Карта отражает не только кластерную структуру данных, но также зависит от...
 - свойств сглаживающего ядра;
 - (случайной) инициализации;
 - (случайного) выбора x_i в ходе итераций.
- **Искажения.** Близкие объекты исходного пространства могут переходить в далёкие точки на карте, и наоборот.

Рекомендуется только для разведочного анализа данных.

47. Ансамбль моделей машинного обучения.

Ансамбль моделей машинного обучения представляет собой методологию, при которой несколько отдельных моделей (так называемых базовых алгоритмов) объединяются в единую систему с целью повышения общей точности и устойчивости

предсказаний. Основная идея ансамблевого подхода заключается в том, что слабые модели, допускающие ошибки на различных подмножествах данных, при объединении в ансамбль способны компенсировать ошибки друг друга, формируя более надёжный и точный итоговый предсказатель. Это особенно актуально в случае нестабильных алгоритмов, чувствительных к колебаниям в данных, таких как деревья решений.

Обучающая выборка: $X^l = (x_i, y_i)$ - набор данных из пространства $X \times Y$, где y_i - истинные ответы. Базовые алгоритмы: $g_t : X \rightarrow Y$, $t = 1, \dots, T$.

Вначале происходит декомпозиция базовых алгоритмов: $g_t(x) = C(b_t(x))$, где b_t - алгоритмические операторы, которые преобразуют входные данные в промежуточное представление, C - решающее правило, которое принимает финальное решение.

В общем виде задача ансамбля описывается в следующем виде:
 $g_t : X \xrightarrow{b_t} R \xrightarrow{C} Y$, где R – пространство оценок. Ансамбль принимает решение на основе оценок, которые дали более слабые алгоритмы, применяя решающее правило.

$$g(x) = C(F(b_1(x), \dots, b_T(x)))$$

где F – агрегирующая (корректирующая) функция.

Общие требования к агрегирующей функции:

- $F(b_1, \dots, b_T, x) \in [\min_t b_t, \max_t b_t]$ – среднее по Коши $\forall x$
- $F(b_1, \dots, b_T, x)$ монотонно не убывает по всем b_t

Примеры агрегирующих функций:

- простое голосование (simple voting):

$$F(b_1, \dots, b_T) = \frac{1}{T} \sum_{t=1}^T b_t$$

- взвешенное голосование (weighted voting):

$$F(b_1, \dots, b_T) = \sum_{t=1}^T \alpha_t b_t, \quad \sum_{t=1}^T \alpha_t = 1, \quad \alpha_t \geq 0$$

- смесь алгоритмов (mixture of experts)

с функциями компетентности (gating function) $G_t : X \rightarrow \mathbb{R}$

$$F(b_1, \dots, b_T, x) = \sum_{t=1}^T G_t(x) b_t(x)$$

48. Методы стохастического ансамблирования.

Методы стохастического ансамблирования представляют собой класс подходов в машинном обучении, при которых разнообразие в ансамбле достигается за счёт введения

случайности в процесс обучения отдельных моделей. Основная идея стохастического ансамблирования заключается в том, что различные варианты исходных данных или их представления позволяют обучить модели, совершающие разные ошибки, и при объединении этих моделей можно получить предсказатель с более высокой обобщающей способностью, чем любая отдельная модель.

1. Bagging (Bootstrap Aggregating). Bagging использует бутстрэп-подвыборки обучающих данных, сформированные случайной выборкой с возвращением (возможны повторы объектов в одной подвыборке). Каждая модель обучается на своей случайной подвыборке. Совокупность таких моделей даёт устойчивый и точный результат. В задачах классификации итоговый ответ принимается путём голосования, в регрессии — усреднением.
2. Pasting. Pasting аналогичен bagging, за исключением одного ключевого различия: подвыборки формируются без возвращения, то есть каждый объект обучающей выборки включается в подвыборку только один раз.
3. Random Subspaces. Метод случайных подпространств предполагает, что каждая модель обучается на всех объектах, но на случайном подмножестве признаков. Таким образом, разнообразие ансамбля обеспечивается за счёт того, что разные модели оперируют с разными проекциями исходного признакового пространства.
4. Random Patches. Метод случайных фрагментов (random patches) объединяет идеи bagging/pasting и random subspaces: каждая модель обучается на случайной подвыборке объектов и признаков одновременно.
5. Cross-Validated Committees Метод комитетов на основе кросс-валидации (cross-validated committees) строится по следующей схеме: проводится k -кратная кросс-валидация, в ходе которой обучаются k моделей на различных разбиениях данных. После завершения обучения модели объединяются в ансамбль, и их предсказания агрегируются (усредняются или выбирается наиболее вероятный класс).

В качестве примера организации ансамблевого алгоритма реализуем bagging + random subspaces.

X^l — обучающая выборка, T — количество базовых алгоритмов в ансамбле, ℓ' — объём обучающих подвыборок (для bagging), n' — размерность признаковых подпространств, ε_1 — порог качества базовых алгоритмов на обучении, ε_2 — порог

качества базовых алгоритмов на контроле. На выходе ожидаем получить базовые алгоритмы $b_t, t = 1, \dots, T$.

1. Определяется случайная подвыборка U_t размерности l^t из X^l для баггинга;
2. G_t – случайное подмножество мощности n' для random subspaces;
3. $b_t = \mu(G_t, U_t)$ – обучение базового алгоритма на подвыборке U_t с использованием только признаков из G_t ;
4. Если $Q(b_t, U_t) > \varepsilon_1$, то не включать b_t в ансамбль (алгоритм не прошёл порог качества на этапе обучения), если $Q(b_t, X^l \setminus U_t) > \varepsilon_2$, то не включать b_t в ансамбль (алгоритм не прошёл порог качества на этапе тестирования);
5. Шаги 1-4 повторяются, пока не сформируются T алгоритмов.

49. Случайный лес.

Обучение случайного леса:

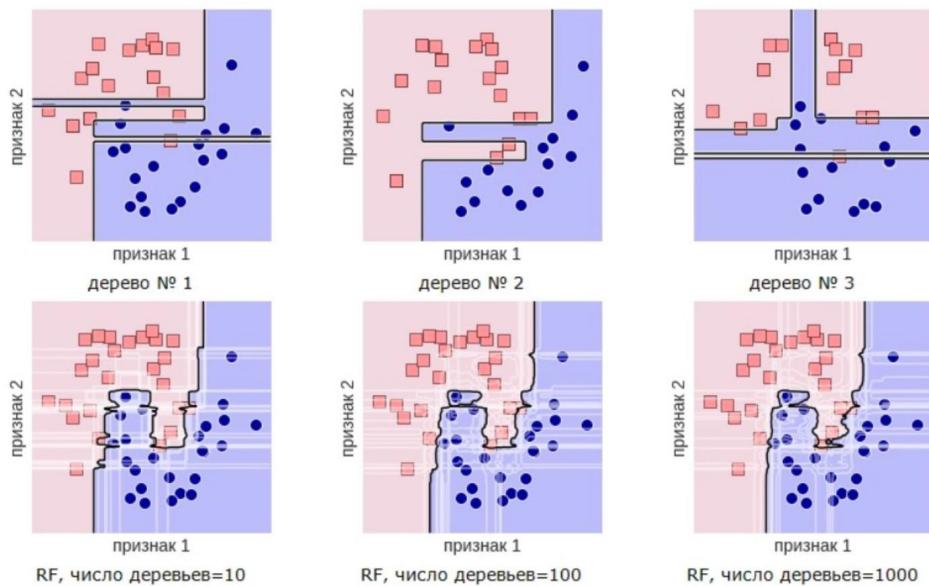
- бэггинг над решающими деревьями, без pruning
- признак в каждой вершине дерева выбирается из случайного подмножества k из n признаков. По умолчанию $k = \lfloor n/3 \rfloor$ для регрессии, $k = \lfloor \sqrt{n} \rfloor$ для классификации

Параметры, которые можно настраивать (в частности, по OOB):

- число T деревьев
- число k случайно выбираемых признаков
- максимальная глубина деревьев
- минимальное число объектов в расщепляемой подвыборке
- минимальное число объектов в листьях
- критерий расщепления: MSE для регрессии, энтропийный или Джини для классификации

Bagging – билет 48, pruning – билет 35, деревья принятия решений – билет 33, MSE – билет 39, энтропийный критерий или Джини – билет 34.

Пример разделения выборки с помощью отдельных деревьев (показаны соответствующие бутстреп-подвыборки) и случайного леса с числом деревьев 10, 100, 1000:



Разновидности случайных решающих лесов

- Случайный лес (Random Forest)
- Использование большого числа простых решающих деревьев в качестве признаков, в любом классификаторе.
- Oblique Random Forest, Rotation Forest
 $f_v(x)$ — линейные комбинации признаков, выбираемые по энтропийному критерию информативности.
- Решающий список из решающих деревьев:
 - при образовании статистически ненадёжного листа этот лист заменяется переходом к следующему дереву;
 - следующее дерево строится по объединению подвыборок, прошедших через ненадёжные листы предыдущего дерева.

50. Отличие между бэггингом и бустингом.

1. Принцип работы и обучение базовых моделей

Бэггинг предполагает параллельное обучение независимых моделей. Каждая модель обучается на случайной подвыборке обучающих данных, созданной методом бутстрэппинга (случайный выбор объектов с возвращением). Все модели равнозначны, и их предсказания агрегируются — например, голосованием (в задачах классификации) или усреднением (в задачах регрессии). Ошибки отдельных моделей не учитываются при обучении других моделей. Бустинг, напротив, реализует последовательное обучение моделей, где каждая новая модель фокусируется на исправлении ошибок, допущенных предыдущими. Объекты, классифицированные неправильно, получают больший вес на следующем шаге, что позволяет последующим моделям уделять им больше внимания.

2. Тип взаимодействия между моделями

В бэггинге модели независимы друг от друга и обучаются параллельно. Это даёт преимущества с точки зрения параллелизации вычислений и устойчивости к переобучению. В бустинге модели зависят друг от друга, и каждая последующая модель вносит корректировки в итоговое решение. Такая зависимость позволяет более точно подстраиваться под сложные паттерны в данных, но одновременно увеличивает риск переобучения при наличии шумов или при недостаточной регуляризации.

3. Устойчивость к переобучению

Бэггинг демонстрирует хорошую устойчивость к переобучению. Бустинг, особенно без регуляризации, может переобучаться, особенно на шумных данных, поскольку стремится точно подогнать каждое наблюдение, включая шумовые или аномальные точки.

51. Алгоритм AdaBoost.

Основная идея алгоритма заключается в построении композиции слабых моделей (чаще всего простых решающих деревьев глубины 1 — т.н. решающих пней), каждая из которых обучается на данных с адаптивно скорректированными весами объектов. На каждом шаге последующая модель сосредотачивается на тех наблюдениях, которые были ошибочно классифицированы предыдущими моделями, тем самым постепенно улучшая точность всей системы.

На вход подаётся обучающая выборка X^l и параметр T .

- Каждому объекту обучающей выборки присваивается вес $w_i := \frac{1}{l}$, это означает, что изначально все объекты одинаково важны для обучения.
- На каждой итерации алгоритма находим слабый классификатор $b_t = \arg \min N(b; W^t)$ N - Взвешенная ошибка - ошибка классификации, где каждый неправильно классифицированный объект вносит вклад пропорционально своему весу. $N(b; W^t) = \sum_{i=1}^l w_i * L[y_i, b(x_i)]$.
- Вычисление коэффициента важности данного классификатора $\alpha_t := \frac{1}{2} \ln \frac{1 - N(b_t; W^t)}{N(b_t; W^t)}$. Этот коэффициент показывает, насколько "хорош" текущий классификатор: если $N(b_t; W^t) \rightarrow 0$ $\alpha_t \rightarrow +$ (очень хороший алгоритм), $N(b_t; W^t) \rightarrow 0,5$ $\alpha_t \rightarrow 0$ (случайное угадывание), $N(b_t; W^t) \rightarrow 1$ $\alpha_t \rightarrow -$ (не правильный классификатор).
- Далее происходит обновление весов объектов $w_i = w_i \exp(-\alpha_t y_i b_t(x_i))$. Если объект классифицирован правильно, то вес объекта уменьшается, если неправильно – увеличивается.
- Нормализация весов, для того, чтобы в сумме веса давали единицу $w_i = \frac{w_i}{\sum_{j=1}^T w_j}$
- Пункты 2-5 повторяются, пока не получится T различных классификаторов.
- На выходе получится T алгоритмов с их весами $\alpha_t b_t$.

Итоговый сильный классификатор строится как взвешенное голосование:

$$H(x) = \operatorname{sign} \sum_{t=1}^T \alpha_t b_t(x) .$$

52. Градиентный бустинг.

Градиентный бустинг (Gradient Boosting) - относится к семейству бустинг-алгоритмов и является развитием идеи AdaBoost с более общей формулировкой и гибкостью.

На вход подаётся обучающая выборка X' и параметр T .

- Сначала зададим начальные предсказания для объектов из обучающей выборки $a_{o,i} := 0$, $i = 1, \dots, l$ (на старте алгоритма у нас еще нет ни одного базового алгоритма, поэтому предсказание для всех объектов инициализируется нулем).

- На каждой итерации строится базовый алгоритм, приближающий антиградиент

$$b_t := \arg \min_{i=1}^l \left(b(x_i) + L'(a_{t-1,i}, y_i) \right)^2, \text{ где } L' - \text{ производная по функции потерь, } a_{t-1,i}$$

- предсказания для объекта x_i после $t-1$ итераций алгоритма. На каждой итерации мы находим такой базовый алгоритм, который аппроксимирует (то есть наилучшим образом отражает) направление наискорейшего убывания функции потерь (антиградиент).

- Определяем вес для полученного алгоритма b_t (задача одномерной минимизации)

$$\alpha_t := \arg \min_{\alpha > 0} \sum_{i=1}^l L(a_{t-1,i} + \alpha b_t(x_i), y_i). \text{ Определяется оптимальный коэффициент } \alpha_t,$$

который определяет, с каким весом нужно добавить новый базовый алгоритм к композиции.

- Так как на предыдущем шаге был добавлен ещё один алгоритм, то необходимо обновить предсказания $a_{t,i} := a_{t-1,i} + \alpha_t b_t(x_i)$.
- Пункты 2-4 повторяются до того момента, пока не получается T алгоритмов b_t .

Стochastic gradient boosting

Идея: при оптимизации b_t и α_t использовать не всю выборку X^ℓ , а случайную подвыборку, по аналогии с бэггингом

Преимущества:

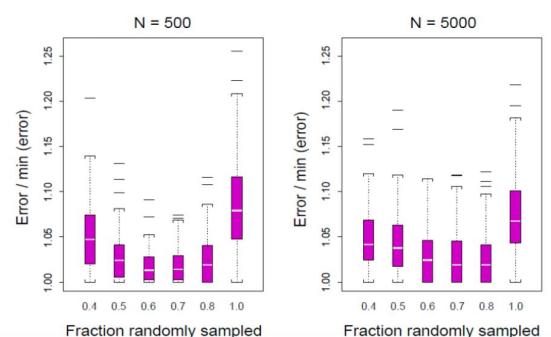
- улучшается сходимость, уменьшается время обучения
- улучшается обобщающая способность ансамбля
- можно использовать несмещённые оценки out-of-bag

Эксперименты:

относительная ошибка при различном объёме выборки N

Вывод:

оптимально сэмплировать около 60–80% выборки



53. Алгоритмы CatBoost, XGBoost.

XGBoost (Extreme Gradient Boosting) представляет собой реализацию градиентного бустинга с рядом инженерных оптимизаций, направленных на повышение скорости и качества. Алгоритм строит ансамбль решающих деревьев, минимизируя регуляризованную функцию потерь, что позволяет бороться с переобучением.

XGBoost добавляет L1 и L2 регуляризацию к целевой функции, что ограничивает сложность моделей и повышает обобщающую способность. Также есть штраф за слишком большой вклад базового алгоритма в результат, так как идея ансамбля строится на совместной работе слабых алгоритмов. Использует предобработку данных, буферизацию и параллельное построение деревьев. Эти и другие методы оптимизации обеспечивает высокую производительность при наличии числовых признаков и табличных данных.

CatBoost (Categorical Boosting) — алгоритм градиентного бустинга, разработанный Яндексом, с упором на обработку категориальных признаков и автоматизацию многих этапов предобработки данных.

В алгоритме CatBoost реализован встроенный механизм преобразования категориальных признаков в числовые (через статистику по обучающей выборке) без необходимости one-hot или label encoding. В классических реализациях градиентного бустинга возникает «предвзятость» оценки градиента, поскольку модель обучается на той же выборке, по которой вычислен градиент. CatBoost устраняет эту проблему с помощью схемы «упорядоченного бустинга», в которой предсказания формируются только на основе предыдущих примеров. CatBoost поддерживает многопоточную и GPU-обработку, что позволяет масштабироваться на большие объёмы данных.

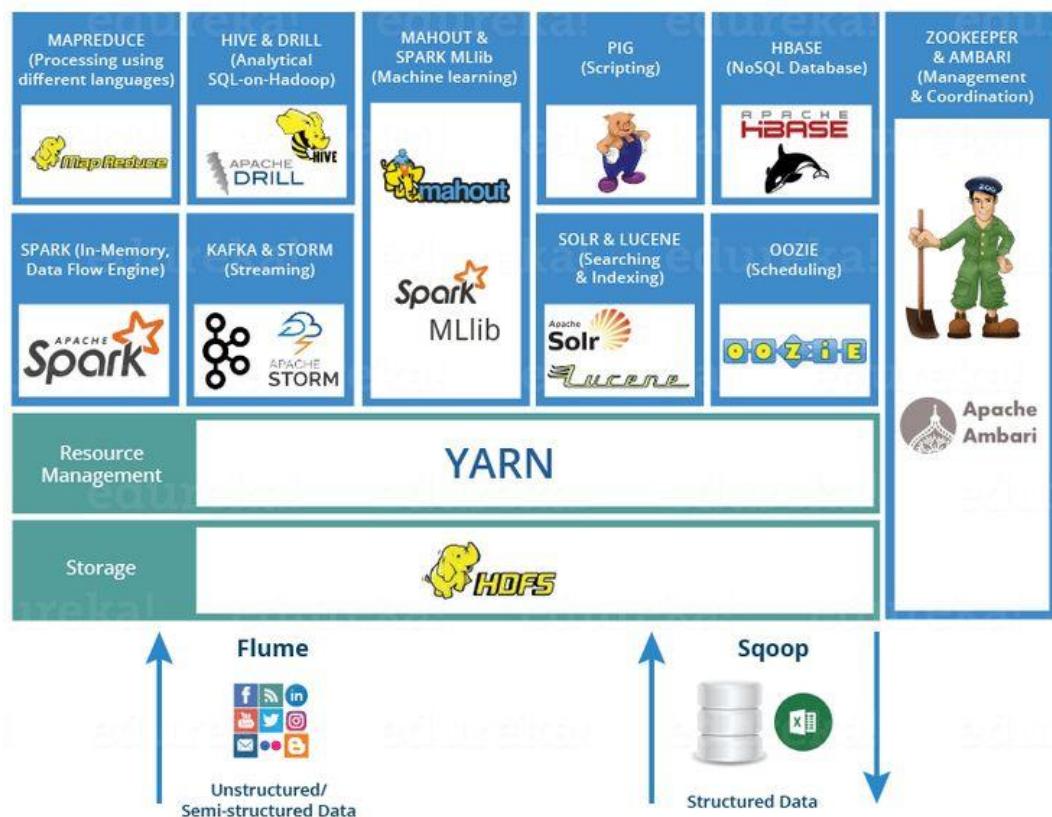
Формально CatBoost минимизирует ту же функцию потерь, что и XGBoost, но за счёт иной организации градиентов и работы с признаками обеспечивает более высокую устойчивость к переобучению и часто лучшее качество при меньшем количестве ручной настройки.

54. Работа с большими данными. Экосистема Apache Hadoop.

Работа с большими данными — это направление в области анализа данных, ориентированное на обработку, хранение и анализ массивов данных объёмом от терабайт

до петабайт и выше, которые не могут быть эффективно обработаны с использованием традиционных реляционных СУБД и однопроцессорных вычислений. Особенности работы с большими данными включают масштабируемость, распределённое хранение, параллельную обработку и устойчивость к отказам.

Экосистема Apache Hadoop — это один из наиболее распространённых инструментов для работы с большими данными. Она представляет собой фреймворк с открытым исходным кодом, обеспечивающий распределённое хранение и параллельную обработку данных на кластерах из обычных серверов.



Основные компоненты Apache Hadoop:

1. HDFS (Hadoop Distributed File System) - распределённая файловая система, предназначенная для хранения огромных объёмов данных. Файлы разбиваются на блоки (по умолчанию 128 МБ или 256 МБ), которые дублируются и распределяются по узлам кластера для обеспечения надёжности и отказоустойчивости. Один узел кластера (NameNode) отвечает за метаинформацию и структуру файловой системы, остальные (DataNode) — за хранение данных.
2. YARN (Yet Another Resource Negotiator) - система управления ресурсами кластера, которая планирует и управляет вычислительными ресурсами для различных приложений.

3. MapReduce - модель программирования для обработки больших наборов данных с использованием различных языков программирования. Map — каждый фрагмент данных обрабатывается локально с целью извлечения ключ-значение пар; Reduce — пары с одинаковыми ключами агрегируются и обрабатываются совместно.
4. Spark - быстрый движок для обработки данных в памяти.
5. Hive & Drill - инструменты для аналитических SQL-запросов поверх больших данных. Hive предоставляет SQL-подобный интерфейс для Hadoop.
6. Pig - высокоуровневый язык сценариев для анализа больших наборов данных.
7. Mahout & Spark MLlib - библиотеки машинного обучения для кластерного анализа и создания рекомендательных систем.
8. Kafka & Storm - платформы для обработки потоковых данных в реальном времени.
9. Solr & Lucene - инструменты для полнотекстового поиска и индексирования документов.
10. HBase - NoSQL база данных, построенная поверх HDFS для быстрого доступа к большим таблицам.
11. Oozie - система планирования и координации рабочих процессов Hadoop.
12. Zookeeper & Ambari - Zookeeper обеспечивает координацию распределенных приложений, а Ambari предоставляет веб-интерфейс для управления и мониторинга кластера Hadoop.
13. Flume - сервис для сбора и агрегации неструктурированных и полуструктурных данных из различных источников (социальные сети, логи, email).
14. Sqoop - инструмент для передачи структурированных данных между Hadoop и реляционными базами данных (например, из Excel в HDFS).

55. Файловая система HDFS.

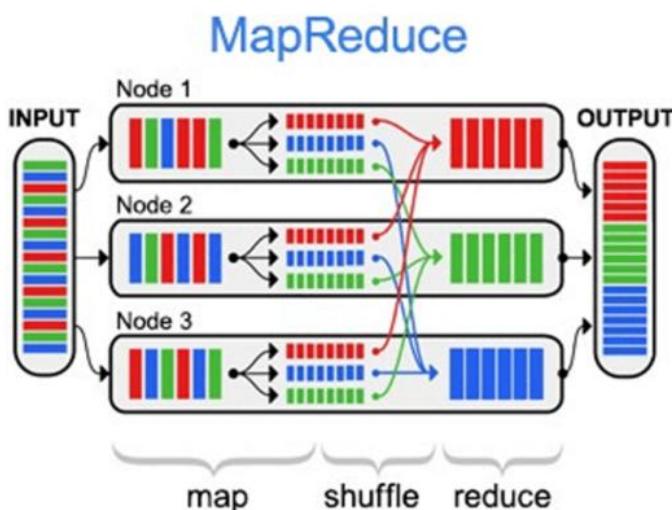
Hadoop Distributed File System (HDFS) представляет собой распределённую файловую систему, разработанную как ключевой компонент экосистемы Apache Hadoop для хранения и управления большими объёмами данных на кластерах из стандартных серверов. Основная цель HDFS — обеспечить надёжное, масштабируемое и высокопроизводительное хранение данных, способное справляться с объемами от терабайт до петабайт и более.

HDFS использует архитектуру «мастер-слейв» и состоит из следующих основных компонентов:

- NameNode — центральный управляющий компонент, отвечающий за хранение метаинформации о файловой системе, включая структуру каталогов, расположение блоков, права доступа, иерархию файлов и т.д. NameNode не хранит сами данные, а управляет распределением блоков по DataNode.
- DataNode — узлы, на которых непосредственно хранятся блоки данных. DataNode обслуживают запросы на чтение и запись блоков, передавая данные клиентам по требованию NameNode.
- Secondary NameNode — выполняет функцию периодического сохранения снимков состояния NameNode (checkpointing) для повышения отказоустойчивости.

Файлы в HDFS делятся на блоки фиксированного размера (обычно 128 МБ или 256 МБ), которые затем распределяются между DataNode. Это позволяет обрабатывать файлы параллельно и эффективно использовать ресурсы кластера. Каждый блок данных хранится в нескольких копиях (по умолчанию 3 реплики) на разных узлах для обеспечения отказоустойчивости. В случае выхода из строя одного из DataNode, данные можно восстановить с другой реплики.

56. Алгоритм Map-Reduce.



Алгоритм MapReduce — это модель программирования и соответствующий фреймворк для распределённой обработки больших объёмов данных на кластере вычислительных узлов. Он был разработан Google и позднее реализован в открытом виде в рамках экосистемы Apache Hadoop.

Алгоритм состоит трёх основных компонентов:

1. Мар-функция — применяется к входным данным и трансформирует каждый элемент во множество пар ключ–значение (key–value). Это позволяет разбить задачу на независимые подзадачи, которые могут быть выполнены параллельно на разных узлах кластера.
2. Shuffle and Sort — промежуточный этап, на котором система автоматически группирует все значения, соответствующие одному и тому же ключу. Shuffle отвечает за передачу данных между узлами, а Sort — за упорядочивание ключей, что позволяет упростить Reduce-фазу.
3. Reduce-функция — агрегирует значения, соответствующие одному ключу, и производит финальный результат. Каждая Reduce-задача работает только с данными одного ключа и выполняется независимо от других.

Пример: необходимо посчитать количество вхождений каждого слова в большом наборе текстов. Сначала мар-функция принимает строку текста, разбивает её на слова и генерирует пары вида (слово, 1) для каждого слова, пары (слово, 1) сортируются и группируются по ключу, reduce-функция для каждого слова суммирует все единицы, чтобы получить итоговое количество вхождений.

57. Apache Spark и распределенные наборы данных (RDD).

Apache Spark представляет собой распределённую вычислительную платформу с открытым исходным кодом, предназначенную для быстрой обработки больших объёмов данных. В отличие от Hadoop MapReduce, Spark поддерживает как пакетную, так и интерактивную обработку, а также предоставляет высокоуровневые API для Java, Scala, Python и R. Одним из ключевых компонентов Spark является концепция распределённых наборов данных — Resilient Distributed Datasets (RDD).

RDD является абстракцией неизменяемой (immutable), распределённой коллекции объектов, которую можно обрабатывать параллельно. Эти коллекции разбиты на логические партиции, которые могут находиться на разных узлах кластера. Основные характеристики RDD:

1. Устойчивость (Resilience). RDD обладает механизмом восстановления после сбоев за счёт линии родословной (lineage). Каждое RDD может быть воссоздано из

исходных данных и последовательности трансформаций, применённых к ним, без необходимости хранения промежуточных результатов.

2. Распределённость (Distributed). Данные хранятся и обрабатываются параллельно на кластере, что обеспечивает высокую производительность и масштабируемость.
3. Ленивая трансформация (Lazy evaluation). Операции над RDD не исполняются немедленно, а откладываются до момента вызова действия (action), что позволяет Spark оптимизировать выполнение и уменьшать количество проходов по данным.
4. Неизменяемость (Immutable). После создания RDD не может быть изменено, любые преобразования создают новое RDD. Это упрощает параллелизм и делает систему более надёжной.

Операции в Spark делятся на два типа:

- Трансформации (Transformations) — ленивые операции, создающие новое RDD из существующего. Примеры: map(), filter(), flatMap(), union(), distinct(), groupByKey(), reduceByKey(), join().
- Действия (Actions) — операции, инициирующие выполнение трансформаций и возвращающие результат. Примеры: collect(), count(), reduce(), first(), take(), saveAsTextFile().

Наименование	Преобразование
map(функция)	Применение функции к каждому элементу исходного RDD
flatMap(функция)	Функция от одного элемента может возвращать последовательность
filter(функция)	Возвращаются те элементы, для которых функция возвращает истину
groupByKey()	$(K, V) \rightarrow (K, \text{seq}(V))$
reduceByKey(функция)	$(K, V) \rightarrow (K, \text{функция}(\text{seq}(V)))$
join(другой RDD)	$(K,V1),(K,V2) \rightarrow (K,(V1,V2))$
cogroup(другой RDD)	$(K,V1), (K,V2) \rightarrow (K, \text{seq } (V1 \text{ seq } (V2)))$

Spark позволяет кэшировать или сохранять RDD в памяти или на диск с помощью cache() и persist(). Это особенно важно при повторном использовании RDD в множественных действиях, так как предотвращает избыточные вычисления. Также Spark поддерживает итеративные алгоритмы, такие как алгоритмы МЛ или графовые вычисления, которые неудобны в MapReduce.

58. Принципы работы рекомендательных систем.

Рекомендательные системы представляют собой класс интеллектуальных систем, предназначенных для персонализированной фильтрации информации. Их основная цель заключается в предоставлении пользователю наиболее релевантных и интересных объектов (товаров, фильмов, статей и т.п.) на основе анализа его предпочтений, поведения и взаимодействий с системой.

Основные компоненты алгоритмов рекомендательных систем: U — все пользователи системы, которым необходимо предоставлять рекомендации, I - все элементы, которые могут быть рекомендованы пользователям.

Основные типы данных:

- Транзакционные данные: $D = (u_t, i_t, y_t)_{t=1}^T \subset U \times I \times Y$. Данные представляют собой описание совершенный транзакций: u_t - пользователь, совершивший транзакцию в момент времени t , i_t - объект, с которым взаимодействовал пользователь в момент времени t , y_t - описание транзакции (оценка, отзыв, характеристики взаимодействия), Y — пространство всех возможных описаний транзакций, T — общее количество транзакций в системе.
- Матрица отношений (или кросс-табуляции): $R = (r_{ui})_{U \times I}$. Это двумерная матрица размерности $|U| \times |I|$, где каждый элемент r_{ui} представляет отношение пользователя u к объекту i . Формируется через агрегацию транзакционных данных $r_{ui} = \text{aggr}\{(u_t, i_t, y_t) \in D \mid u_t = u, i_t = i\}$. Функция aggr агрегирует все транзакции конкретного пользователя u с конкретным объектом i . $r_{ui} \in \{0, 1\}$ - бинарные данные, которые показывают взаимодействовал или нет пользователь u с объектом i . $r_{ui} \in \{1, \dots, M\}$ - порядковые значения, представляющие степень предпочтения, например оценки от 1 до 5 звёзд и тп.

Основные задачи в рекомендательных системах:

- Заполнение пропусков в ячейках r_{ui} . Это задача предсказания неизвестных значений r_{ui} в матрице отношений. Поскольку большинство пользователей взаимодействует лишь с небольшой частью доступных объектов, матрица R

обычно очень разрежена. Алгоритм должен предсказать, какую оценку пользователь и поставил бы объекту i , если бы с ним взаимодействовал.

- Ранжирование top-n рекомендаций. Это задача формирования упорядоченного списка из n наиболее релевантных объектов для конкретного пользователя u , или для конкретного объекта i .

U — пользователи Интернет

I — текстовые документы (сайты/страницы/новости и т.п.)

W — словарь слов (токенов/термов), образующих документы

r_{ui} = [пользователь u посетил/лайкнул документ i]

n_{iw} = частота слова w в документе i

Web Content Mining — анализ данных о контенте (n_{iw})

Web Usage Mining — анализ данных об использовании (r_{ui})

Основная гипотеза WUM: действия пользователя характеризуют его интересы, возможности, привычки, вкусы

Задачи персонализации: найти релевантные документы i для пользователя, документа или подборки документов

Примеры: Я.Музыка, YouTube, Дзен, МирТесен, SurfingBird.ru

59. Коллаборативная и контентная фильтрация.

Коллаборативная фильтрация основывается на анализе взаимодействий между пользователями и объектами (например, просмотренных фильмов, оценок товаров, покупок). Основной принцип — пользователи с похожими интересами в прошлом будут иметь схожие предпочтения в будущем.

- User-based (по пользователям): если пользователь А схож с пользователем В, и В высоко оценил объект X, то X рекомендуется А.
- Item-based (по объектам): если пользователю А понравился объект X, и многим другим пользователям, которым нравился X, также нравился Z, то Z рекомендуется А.

Метод оперирует исключительно матрицей пользователь–объект, без знания характеристик объектов или профилей пользователей.

Контентная фильтрация использует информацию о свойствах объектов и предпочтениях пользователей. Система создает профиль пользователя, исходя из характеристик объектов, с которыми он взаимодействовал, и ищет объекты с похожими

свойствами. Например, если пользователь предпочитает комедии с актёром N, система будет рекомендовать фильмы с аналогичным жанром или актёром.

Для представления объектов и пользователей используют признаки (жанр, категория, цена, ключевые слова), и на их основе рассчитывается схожесть (например, через метод ближайших соседей).

60. Техники коллаборативной фильтрации: memory-based и model-based.

Методы memory-based не требуют предварительного обучения модели. Вместо этого они используют всю матрицу взаимодействий между пользователями и объектами напрямую. Рекомендации формируются на основе сходства между пользователями или объектами, измеренного по историческим данным. (билет 61).

Выделяют два подтипа memory-based фильтрации:

- User-based collaborative filtering — метод, при котором для каждого пользователя находятся другие пользователи с похожими оценками или поведением. (см билет 59).
- Item-based collaborative filtering — метод, при котором для каждого объекта определяются другие объекты, получившие похожие отклики от пользователей. (см билет 59).

Преимуществами memory-based методов являются простота реализации и интерпретируемость результатов. Однако они страдают от проблемы масштабируемости, чувствительности к разреженности данных

Методы model-based предполагают построение обучаемой модели на основе существующих взаимодействий между пользователями и объектами. Такие модели способны выявлять скрытые зависимости и использовать их для генерации рекомендаций.

К наиболее распространённым методам model-based относятся

- Матричная факторизация — техника, при которой исходная матрица пользователь-объект представляется как произведение двух матриц меньшей размерности, соответствующих латентным (скрытым) представлениям пользователей и объектов. Эти признаки не даны явно, их значения вычисляются автоматически в процессе

обучения модели таким образом, чтобы произведение соответствующих векторов (скалярное произведение латентных представлений пользователя и объекта) хорошо приближало реальные оценки. (бillet 62).

- Нейронные сети — современные подходы, использующие глубокое обучение, включая архитектуры типа Neural Collaborative Filtering, которые обучаются напрямую на пользовательских откликах и предоставляют высокоточную модель рекомендаций.
- Градиентный бустинг и деревья решений.

Model-based методы отличаются высокой точностью и устойчивостью к разреженности данных. Они позволяют использовать дополнительные контекстные признаки (время, геолокацию, устройство и пр.), но требуют больших вычислительных ресурсов и наличия этапа обучения. Кроме того, такие модели сложнее в интерпретации.

61. Корреляционные модели в коллаборативной фильтрации. Непараметрическая регрессия, функции сходства.

Корреляционные модели в коллаборативной фильтрации относятся к классу методов, основанных на поиске сходства между пользователями или объектами, исходя из их поведения, без использования дополнительных метаданных. Основная идея таких моделей заключается в том, что пользователи, проявляющие сходные оценки в прошлом, вероятно, будут согласны и в будущем, а объекты, получающие схожие оценки от одних и тех же пользователей, вероятно, обладают схожими характеристиками.

В корреляционных моделях центральным элементом является функция сходства, с помощью которой измеряется степень близости между двумя пользователями или двумя объектами. Наиболее распространёнными функциями сходства являются:

- Корреляция Пирсона: $S(u, v) = \frac{\sum_{i \in I(u,v)} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I(u,v)} (r_{ui} - \bar{r}_u)^2} \sqrt{\sum_{i \in I(u,v)} (r_{vi} - \bar{r}_v)^2}}$, где

r_{ui}, r_{vi} - оценки объекта i от пользователей u и v , \bar{r}_u, \bar{r}_v - средние оценки u и v ,

$I(u,v)$ - множество объектов, которые оценили оба пользователя. Данная метрика оценивает насколько линейно связаны два вектора, измеряя степень совместного отклонения от их средних значений. Изменяется в пределах [-1.0;1.0].

- Косинусная мера сходства: $S(u, v) = \frac{\mathbf{r}_u \cdot \mathbf{r}_v}{\|\mathbf{r}_u\| \|\mathbf{r}_v\|} = \frac{\sum_{i \in I(u,v)} r_{ui} * r_{vi}}{\sqrt{\sum_{i \in I(u)} r_{ui}^2} \sqrt{\sum_{i \in I(v)} r_{vi}^2}}$.

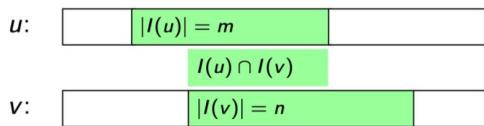
Измеряет угловую близость между двумя векторами в пространстве, то есть насколько направлены векторы одинаково, без учёта их длины. Изменяется в пределах [-1.0;1.0].

Аналогичные функции сходства можно вычислить для пар объектов.

Задача функции сходства для бинарных данных заключается в определении, сколько общих объектов оценили пользователи u и v , то есть насколько близки пользователи.

Функции сходства для бинарных данных

Чем больше $|I(u) \cap I(v)|$, тем более схожи клиенты u и v :



Мера близости Жаккара (Jaccard similarity):

$$S(u, v) = \frac{|I(u) \cap I(v)|}{|I(u) \cup I(v)|}$$

Точный тест Фишера (Fisher's Exact Test, FET)

Вероятность пересечения оценок при нулевой гипотезе, что клиенты u и v совершают свой выбор независимо:

$$S(u, v) = -\log P\{|I(u) \cap I(v)| = i\} = -\log \frac{C_m^i C_{n-i}^{n-m}}{C_n^n}$$

Непараметрическая регрессия — это метод прогнозирования (задача заполнения пропусков, см билет 58), который не предполагает конкретную параметрическую форму зависимости между переменными. Вместо этого он использует локальное взвешивание по сходству для восстановления неизвестных значений.

- Оценка рейтинга по схожим пользователям (User-Based Collaborative

Filtering): $\hat{r}_{ui} = \bar{r}_u + \frac{\sum_{v \in U(u)} S(u, v) (r_{vi} - \bar{r}_v)}{\sum_{v \in U(u)} S(u, v)}$, где \hat{r}_{ui} - предсказанный рейтинг

пользователя u для объекта i , \bar{r}_u - средний рейтинг пользователя u по всем его оценкам, $U(u)$ - коллаборация, множество клиентов, схожих с u . Метрика сначала

учитывает базовые (средние) оценки пользователя u , потом отклонения похожих пользователей, насколько пользователь v оценил объект i выше/ниже своего среднего уровня.

- Оценка рейтинга по сходим объектам (Item-Based Collaborative Filtering):

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in I(i)} S(i, j)(r_{uj} - \bar{r}_j)}{\sum_{j \in I(i)} S(i, j)}.$$
 По сути та же метрика, только опирается на оценку схожих объектов, а не на оценки схожих пользователей.

62. Понятие латентной модели.

Латентная модель — это математический подход в машинном обучении и статистике, который предполагает существование скрытых (латентных) факторов, объясняющих наблюдаемые данные.

Суть алгоритма: по данным D (Транзакционные данные, билет 58) оцениваются векторы $(p_{tu})_{t \in G}$ вектор (каждый элемент p_{tu} показывает, насколько сильно клиент u связан с латентным фактором t . Например, если t представляет "любовь к боевикам", то p_{tu} может быть числом от 0 до 1, показывающим степень этой любви у пользователя u) и $(q_{ti})_{t \in H}$ вектор (аналогично, связь объекта i и латентным фактором t). $|G|$ — количество латентных факторов для пользователей (например, 50 скрытых предпочтений), $|H|$ — количество латентных факторов для объектов (например, 50 скрытых характеристик), $|U|$ — общее количество пользователей (например, 1 миллион), $|I|$ — общее количество объектов. $|G| \ll |U|$ и $|H| \ll |I|$, что означает, что количество латентных факторов значительно меньше количества пользователей.

Ко-кластеризация разделяет данные на группы:

- Каждый клиент/объект принадлежит строго одному кластеру (Жёсткая кластеризация);
- Клиенты/объекты могут принадлежать нескольким кластерам с разными степенями принадлежности (мягкая кластеризация).

Матричные разложения ($G \equiv H$) восстанавливают исходную матрицу через произведение низкоранговых матриц. Это позволяет выявить скрытые паттерны в данных пользователь-объект.

Вероятностные модели моделируют p_{tu} и q_{ti} как условные вероятности $p(t|u)$ и $q(t|i)$, где t представляет латентные темы или факторы.

63. Матричные разложения. Сингулярное разложение.

Матричные разложения заключаются в представлении исходной матрицы в виде произведения нескольких более простых матриц, обладающих особыми свойствами, что позволяет упростить анализ, хранение и обработку данных.

Одним из наиболее важных и универсальных видов матричных разложений является сингулярное разложение (SVD, Singular Value Decomposition). Оно применяется как к квадратным, так и к прямоугольным матрицам.

Постановка задачи: $\|R - PQ^T\|^2 \rightarrow \min_{P,Q}$ означает, что необходимо найти такие P и Q , при которых минимизируется квадрат нормы Фробениуса (корень из суммы квадратов элементов) разницы между исходной матрицей $R_{(m \times n)}$ и произведением PQ^T .

Сингулярное разложение: $\tilde{R} = \underbrace{V}_{P} \underbrace{\sqrt{D}}_{Q^T} \underbrace{U^T}_{Q}$, где \tilde{R} - аппроксимация исходной матрицы R , V и U – ортогональные матрицы ($V^T V = I, U^T U = I$), D - диагональная матрица сингулярных значений (по диагонали матрицы D в результате решения будут сингулярные значения, расположенные по важности от самого значимого значения до наименее значимого). Сжатие информации достигается за счёт приближённого восстановления исходной матрицы с использованием лишь первых k сингулярных значений:

$$R_k = U_k D_k V_k^T$$

где $k = \min(m, n)$.

В контексте задачи рекомендаций, в качестве примера R – матрица оценок разных фильмов разными пользователями:

- Матрица V - базисные направления в пространстве признаков, они представляют направления данных, по которым различаются объекты (тут V может быть латентными признаками фильмов, например, жанр, стиль и тп).

- Матрица D – содержит сингулярные значения, которые показывают степень важности направления (чем больше направление из матрицы V , тем более различимые признаки, а значит более значимые). Не значимые признаки можно отбросить, поэтому берётся матрица ранга k .
- Матрица U - представление объектов в пространстве сингулярных компонент. Произведение $U_k D_k$ можно интерпретировать как новые координаты объектов в пространстве пониженной размерности. Каждый вектор u_i показывает, насколько сильно объект выражает i -й латентный признак из матрицы V .

64. Измерение качества рекомендаций.

Измерение качества рекомендаций

Многокритериальность в рекомендательных системах:

- *Разнообразие* (diversity): число рекомендаций из разных категорий или степень различия рекомендаций между сессиями клиента
- *Новизна* (novelty): сколько среди рекомендаций объектов, новых для данного клиента
- *Покрытие* (coverage): доля объектов, которые хотя бы раз побывали среди рекомендованных
- *Догадливость* (serendipity): способность угадывать неожиданные нетривиальные предпочтения клиентов
- *Предвзятость* (propensity): прошлые рекомендации влияют на выбор клиентов, смешая последующие рекомендации

Можно оптимизировать линейную комбинацию критериев, либо оптимизировать один при ограничениях на остальные.

- RMSE – точность рекомендации (как в NetflixPrize (Открытое соревнование на лучший алгоритм предсказания оценки, которую зритель поставит фильму, на основе предыдущих оценок этого и других зрителей))): $RMSE^2 = \frac{(r_{ui} - \hat{r}_{ui})^2}{(u,i) \in D}$.

RMSE измеряет среднеквадратичное отклонение предсказанных рейтингов от реальных. Чем меньше RMSE, тем точнее система предсказывает рейтинги. Точность предсказаний не гарантирует хороших рекомендаций - система может

точно предсказать, что пользователь поставит фильму 3 звезды, но это не означает, что этот фильм стоит рекомендовать

- Precision@k: $\frac{|R_u(k) \cap L_u|}{|R_u(k)|}$, где $R_u(k)$ — первые k рекомендованных товаров для пользователя u , L_u — товары, которые пользователь u реально предпочел (купил, высоко оценил).
- Recall@k: $\frac{|R_u(k) \cap L_u|}{|L_u|}$. Recall показывает, какую долю от всех релевантных товаров система смогла найти в топ- k .
- Метрики для ранжирования: MAP (Mean Average Precision) — учитывает позицию релевантных элементов в списке рекомендаций, NDCG (Normalized Discounted Cumulative Gain) — учитывает как релевантность, так и позицию в рейтинге, с убывающим весом для низких позиций.

65. Понятие искусственного нейрона.

Искусственный нейрон является базовым элементом искусственной нейронной сети и представляет собой математическую модель, вдохновлённую работой биологического нейрона. Цель модели — эмулировать механизм обработки и передачи информации в мозге, обеспечивая способность к обучению, обобщению и принятию решений на основе входных данных.

Искусственный нейрон принимает на вход вектор параметров $X = \{x_1, x_2, \dots, x_n\}$, который соответствует наблюдению в пространстве признаков. Каждый входной сигнал умножается на собственный вес w_i отражающий значимость данного признака (данний участок символизирует синапсы между нейронами). Далее вычисляется взвешенная сумма всех входов с добавлением смещения (свободного члена) b . Это выражается формулой:

$$z = \sum_{i=1}^n w_i x_i + b$$

Значение z далее передаётся через нелинейную функцию активации $\phi(z)$, которая определяет выход нейрона:

$$y = \phi(z)$$

Функция активации играет ключевую роль в обеспечении способности нейросети аппроксимировать сложные нелинейные зависимости. Классическими примерами таких функций являются сигмоида, гиперболический тангенс, ReLU (Rectified Linear Unit), и её модификации (Leaky ReLU, ELU и другие).

Совокупность нейронов, соединённых по определённой топологии, образует нейронную сеть, способную решать задачи классификации, регрессии, кластеризации и генерации данных.

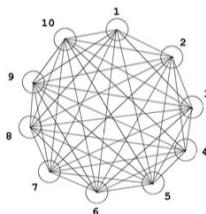
Обучение искусственного нейрона осуществляется путём настройки весов и смещения по методу обратного распространения ошибки на основе градиентного спуска, минимизируя функцию потерь между предсказанием и истинным значением.

66. Сеть Хопфилда.

Сеть Хопфилда (Hopfield Network) - это один из классических типов рекуррентных нейронных сетей, разработанный Джоном Хопфилдом в 1982 году.

Сеть имеет полносвязную архитектуру, каждый нейрон соединен с каждым другим нейроном в сети. Если в сети N нейронов, то количество связей равно $N(N-1)/2$ (см картинку). Вес связи от нейрона i к нейрону j равен весу связи от j к i ($w_{ij} = w_{ji}$). Это ключевое свойство обеспечивает стабильность сети. Нейроны могут находиться только в двух состояниях - активном (+1) или неактивном (-1). В сети Хопфилда все нейроны одновременно являются и входными, и выходными. Это гомогенная структура, где каждый нейрон получает сигналы от всех остальных нейронов, передает свой сигнал всем остальным нейронам и может изменять свое состояние на основе получаемых сигналов.

- Один из видов рекуррентных сетей
 - Полностью рекуррентный
 - Веса симметричны
 - Узлы могут быть только в состояниях *вкл* или *выкл*
 - Случайный порядок перехода от одного узла к другому



- Обучение: **Правило Хебба** (cells that fire together wire together)
- Может вспомнить и восстановить образ по его искаженной или неполной версии

→ ассоциативная память



Сеть обучается по правилу Хебба (нейроны, которые активируются вместе, связываются вместе).

- Правило Хебба (ассоциативное обучение):
 - одновременная активация клеток приводит к выраженному увеличению синаптической силы между этими клетками
- Инициализация весов в дискретной модели Хопфилда:

$$\theta_{ji} = w_{ji} = \frac{1}{N} \sum_{k=1}^{\ell} x_{k,j} x_{k,i}$$

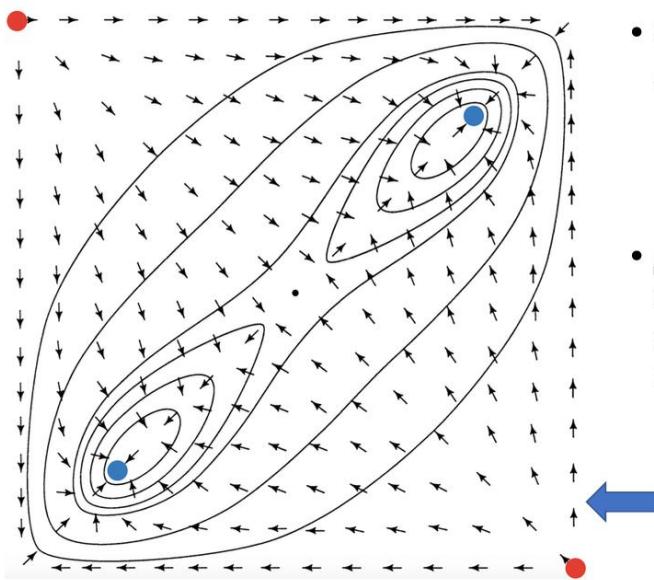
где $x_{k,j}$ – j -ый элемент бинарного вектора \mathbf{x}_k из обучающей выборки

Сеть может запомнить несколько образов (паттернов) и затем восстанавливать их по неполной или искаженной информации.

Принцип работы сети: на вход (в каждый нейрон) поступает информация, например, пиксели искажённой фотографии или фрагмент картинки. На каждом шаге случайно выбирается один нейрон и пересчитывает свое состояние. Процесс продолжается до тех пор, пока состояния нейронов не перестанут изменяться. Финальное состояние всей сети и есть результат.

Энергетический ландшафт сети Хопфилда показывает несколько ключевых аспектов работы системы: каждая точка на плоскости - одно возможное состояние всей сети, высота рельефа - энергия этого состояния (по формуле Ляпунова), низины (синие точки) - состояния с минимальной энергией (сюда сходится система из окружающих областей, обычно соответствуют запомненным образам), возвышенности (красные точки) - состояния с высокой энергией (начальные состояния, искаженные или неполные

входные образы). Формула Ляпунова вычисляет "энергию" любого состояния сети: w_{ij} - вес связи между нейронами i и j , y_i , y_j - состояния нейронов (+1 или -1), минус перед формулой означает, что мы ищем минимумы. Стрелки показывают направление изменения состояния сети: каждая стрелка показывает в какую сторону "потечет" система из данной точки, все стрелки направлены вниз по склону (к уменьшению энергии), система никогда не может подняться вверх по энергии. По сути, обучение сети Хопфилда - создание нужных "ям" в энергетическом ландшафте, а восстановление образа — это скатывание шарика в ближайшую яму.



- **Функция энергии (Ляпунова) дискретной сети:**

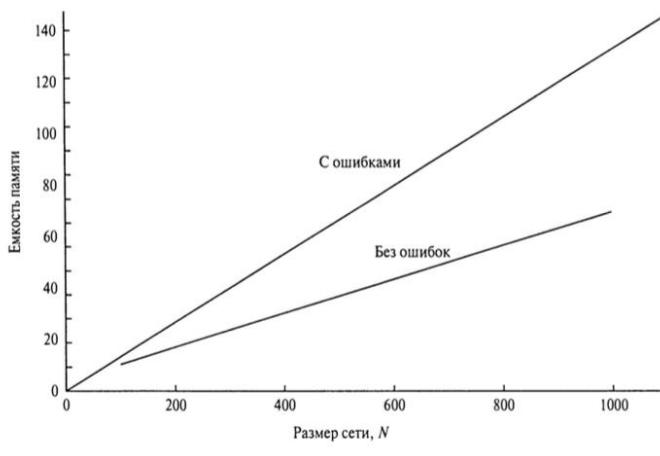
$$E = -\frac{1}{2} \sum_{i=1}^N \sum_{j=1, j \neq i}^N w_{ji} y_i y_j$$

- **Динамика сети Хопфилда описывается механизмом, обеспечивающим нахождение минимумов E .**

Контурная карта энергии непрерывной модели Хопфилда с двумя нейронами и двумя устойчивыми состояниями. Ординаты и абсциссы являются выходами двух нейронов. Устойчивые состояния соответствуют минимумам энергии и показаны синим, неустойчивые экстремумы — красным.

Сеть Хопфилда не может запомнить произвольно большое количество образов. На графике ниже представлены две прямые: максимальное количество образов, которое теоретически можно запомнить (с ошибками) и количество образов для надежного восстановления без ошибок. Перегрузка памяти начинается, когда $\frac{l}{N} > 0.14$, где l - количество запомненных образов, N - количество нейронов в сети. При превышении этого соотношения восстановление становится невозможным, система входит в хаотический режим. $l = 0.14N$ - граница надежности, она означает, что сеть может запомнить 14% изображений от количества нейронов, при этом пороге еще возможно восстановление, но с ошибками. $l = \frac{N}{2 \ln N}$ - практический предел, при нём сеть может восстановить l образов почти без ошибок. Каждый новый запомненный образ накладывается на уже существующие, создавая шумы.

Ёмкость памяти сети Хопфилда



- Восстановление образов из памяти невозможно, если:
$$\frac{\ell}{N} > 0.14$$
- Восстановить ℓ образов с ошибками можно, если:
$$\ell = 0.14N$$
- Восстановить ℓ образов почти без ошибок:
$$\ell = \frac{N}{2 \ln N}$$

67. Понятие стохастического нейрона. Машина Больцмана.

Стохастический нейрон — это элемент искусственной нейронной сети, поведение которого моделируется вероятностным образом. В отличие от классических детерминированных нейронов, где активация определяется жёсткой функцией активации (например, сигмоидой или ReLU), стохастический нейрон активируется с определённой вероятностью, зависящей от взвешенной суммы входных сигналов.

Активация стохастического нейрона обычно описывается распределением Бернулли, где вероятность активации определяется, например, с помощью логистической функции: $P(\text{активация}) = \sigma(w^T x + b)$, где σ — сигмоида ($\sigma(x) = \frac{1}{1 + e^{-x}}$), $w^T x + b$ — линейная комбинация входных данных и смещения. Один и тот же вход может вызывать различную активацию на разных запусках модели, что позволяет таким нейронам моделировать вероятностную природу процессов.

Машина Больцмана (Boltzmann Machine) — это тип стохастической генеративной нейронной сети, построенной на основе стохастических нейронов. Она представляет собой неориентированный граф, в котором нейроны (узлы) соединены симметричными связями (весами).

Цель машины Больцмана — находить такие конфигурации нейронов (то есть такие значения вектора активаций), которые минимизируют специальную энергетическую функцию, называемую энергетической функцией системы.

В классической машине Больцмана энергетическая функция $E(v)$ задаётся формулой:

$$E(v) = - \sum_{i < j} v_i w_{ij} v_j - \sum_i b_i v_i$$

где $v_i \in \{0;1\}$ - состояние нейрона, w_{ij} - симметричный вес между нейронами i и j , b_i - смещение i -го нейрона.

В обучении машины Больцмана цель — настроить веса и смещения так, чтобы конфигурации, соответствующие данным, имели низкую энергию, а нехарактерные — высокую. Энергетическая функция определяет пейзаж вероятностей — где "ямы" (низкая энергия) — это вероятные, "знакомые" конфигурации, а "холмы" (высокая энергия) — маловероятные. Задача обучения — научить сеть распознавать закономерности и структуры в данных.

68. Алгоритм имитации отжига.

Алгоритм имитации отжига (simulated annealing) представляет собой вероятностный эвристический метод глобальной оптимизации, используемый для поиска приближённого глобального экстремума функции в условиях большой размерности пространства решений и наличия множества локальных экстремумов.

Идея алгоритма заключается в управляемом исследовании пространства возможных решений с возможностью принимать временно худшие решения, чтобы избежать попадания в локальные минимумы.

Алгоритм начинается с выбора начальной точки в пространстве решений (множество всех возможных конфигураций (вариантов), которые можно получить в задаче оптимизации, каждая точка в этом пространстве — это конкретное решение задачи, для которого можно вычислить значение целевой функции) и инициализации температуры — положительного числа, определяющего вероятность принятия ухудшающих шагов. Далее на каждом шаге осуществляется переход к соседнему

состоянию по определённому правилу. Если значение целевой функции (энергии) в новом состоянии меньше, чем в текущем, то переход совершается. Если оно больше, то переход совершается с некоторой вероятностью, определяемой выражением: $P = \exp -\frac{\Delta E}{T}$, где ΔE - разность значений функции между новым и текущим состоянием, T - текущая температура.

Преимуществами метода являются простота реализации, устойчивость к попаданию в локальные минимумы и способность справляться с задачами, в которых градиентные методы неприменимы. К недостаткам относится высокая чувствительность к параметрам охлаждения и большое время вычислений при слишком медленном охлаждении.

69. Ограниченная машина Больцмана. Алгоритм контрастного расхождения (contrastive divergence).

Ограниченная машина Больцмана (Restricted Boltzmann Machine, RBM) представляет собой стохастическую генеративную нейронную сеть, состоящую из двух слоёв: видимого слоя (visible layer), который принимает на вход наблюдаемые данные, и скрытого слоя (hidden layer), который отвечает за формирование латентных признаков. Главное ограничение RBM заключается в том, что нейроны внутри одного слоя не соединены между собой, а связи существуют только между видимыми и скрытыми нейронами.

Каждая конфигурация состояний видимых и скрытых узлов обладает определённой энергетической функцией, значение которой указывает на "правдоподобие" данной конфигурации. Цель обучения RBM — минимизация этой энергии для конфигураций, близких к наблюдаемым данным, и увеличение энергии для маловероятных или искусственных конфигураций. Формально энергетическая функция задаётся как:

$$E(v, h) = - \sum_i v_i h_j w_{ij} - \sum_i b_i v_i - \sum_j c_j h_j$$

где v_i, h_j - бинарные состояния видимого и скрытого узлов соответственно, w_{ij} - веса связей между узлами, b_i, c_j - смещения.

Обучение RBM направлено на максимизацию правдоподобия обучающей выборки, что эквивалентно минимизации функции потерь, основанной на разнице между эмпирическим распределением и модельным распределением.

Алгоритм контрастного расхождения (Contrastive Divergence, CD) был предложен для приближённого обучения RBM. Он позволяет эффективно обновлять веса, приближая градиент функции правдоподобия. Основная идея заключается в следующем: видимые узлы инициализируются значениями из обучающей выборки, скрытые узлы активируются на основе текущих значений видимых узлов, на основе скрытых узлов восстанавливаются значения видимых узлов, на основе реконструированных видимых узлов повторно активируются скрытые узлы, градиент рассчитывается как разность между попарными произведениями исходных видимых и скрытых состояний и реконструированных значений.

70. Глубокие сети на основе машины Больцмана.

Глубокие сети на основе машины Больцмана, или глубокие сети доверия (Deep Belief Networks, DBN), представляют собой иерархическую композицию ограниченных машин Больцмана (Restricted Boltzmann Machines, RBM), где каждый уровень сети обучается последовательно и без учителя.

Типичная DBN включает несколько слоёв скрытых узлов, каждый из которых представляет собой RBM. Первый слой принимает на вход наблюдаемые данные (видимый слой), а каждый следующий слой обучается на выходах предыдущего скрытого слоя. Таким образом, сеть постепенно выучивает всё более абстрактные признаки.

Обучение DBN происходит в два этапа: предобучение без учителя и досовершенствование с учителем (fine-tuning).

Предобучение (unsupervised pretraining): Каждый RBM обучается последовательно на выходе предыдущего слоя с помощью алгоритма контрастного расхождения (бillet 69).

Досовершенствование (supervised fine-tuning): После предобучения можно добавить выходной слой (например, softmax) и обучить всю сеть как сверточную или полносвязную с учителем, используя градиентный спуск или backpropagation. Цель —

адаптировать признаки, извлечённые без учителя, под конкретную задачу классификации или регрессии.

DBN способны автоматически извлекать многослойные и абстрактные признаки из необработанных данных, они менее подвержены переобучению благодаря поэтапному обучению, используются в задачах, где сложно получить большое количество размеченных данных, поскольку основная часть обучения происходит без учителя.

Недостатками алгоритма являются высокая вычислительная сложность при глубокой архитектуре, трудности в интерпретации латентных признаков, постепенно вытесняются современными архитектурами, такими как сверточные и трансформерные сети, особенно при наличии большого количества данных.