

1. Что такое машинное обучение? Кто такой Data Scientist? Как связаны ML, DS и AI? (на русском)

Машинное обучение (ML):

Это раздел искусственного интеллекта, который позволяет системам **извлекать закономерности из данных** и делать прогнозы или принимать решения **без явного программирования**. Алгоритмы улучшаются по мере накопления опыта.

Data Scientist (специалист по данным):

Это эксперт, который находит полезные знания в больших объёмах данных. Он сочетает навыки программирования, статистики, машинного обучения и понимания предметной области, чтобы решать задачи и помогать бизнесу принимать решения.

Связь между ML, Data Science и AI:

- **AI** — общее направление по созданию "умных" машин
- **ML** — подмножество AI, связанное с обучением на данных
- **Data Science** — более широкая область: включает ML, анализ, визуализацию и интерпретацию данных

Вместе: ML даёт алгоритмы, Data Science — анализ и решения, а AI — интеллектуальное поведение.

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

AI делится по уровню возможностей и обобщения знаний:

- **ANI (Искусственный узкоспециализированный интеллект):**
Выполняет одну конкретную задачу (например, распознавание лиц, чат-боты).
Это текущий уровень ИИ.
- **AGI (Искусственный общий интеллект):**
Теоретическая модель ИИ, обладающая **человеческим уровнем интеллекта** во всех сферах. Может обучаться и решать любые задачи как человек.
- **ASI (Искусственный сверхинтеллект):**
Гипотетический ИИ, превосходящий человека по разуму, креативности и эмоциям.
- **Слабый ИИ:**
Узкозаточенные системы без самосознания. Выполняют конкретные функции.
- **Сильный ИИ:**
Потенциально сознательная система, способная к мышлению, самообучению и самопониманию, как человек.

Вывод: сегодня человечество использует только ANI. AGI и ASI пока существуют лишь в теории и научной фантастике.



3. История развития ИИ, МО и глубокого обучения (на русском)

- 1950–60-е:
Рождение ИИ. Алан Тьюринг предложил **тест Тьюринга**. Разрабатывались первые логические ИИ-системы.
- 1970–80-е:
Популярны **экспертные системы** на базе правил. Появились первые алгоритмы машинного обучения (персептрон, решающие деревья).
Начался «зимний период ИИ» — снижение интереса из-за ограниченного прогресса.
- 1990–2000-е:
Подъём **машинного обучения**: SVM, случайные леса, байесовские модели.
В 1997 году Deep Blue обыграл чемпиона мира по шахматам.
- 2010–наши дни:
Революция **глубокого обучения**: нейросети с многими слоями. Прорывы в распознавании изображений (AlexNet, 2012), речи, NLP (BERT, GPT), генеративных моделях (GAN, диффузионные модели).

4. Области применения машинного обучения + примеры

Машинное обучение используется в самых разных сферах для решения прикладных задач:

- **Медицина**: прогноз заболеваний, анализ снимков (например, выявление опухолей)
- **Финансы**: обнаружение мошенничества, скоринг клиентов, автоматическая торговля
- **Ритейл**: рекомендации товаров, сегментация клиентов, прогноз спроса
- **Транспорт**: оптимизация маршрутов, автопилоты
- **Обработка языка (NLP)**: чат-боты, машинный перевод, анализ тональности
- **Промышленность**: предиктивное обслуживание оборудования, контроль качества
- **Кибербезопасность**: обнаружение аномалий, классификация вредоносного ПО

Примеры задач:

- Прогнозирование стоимости жилья по району и характеристикам
- Определение спам-писем
- Рекомендация фильмов на основе вкусов пользователя
- Обнаружение подозрительных транзакций в банке

Вывод: машинное обучение — универсальный инструмент, который адаптируется к любому виду данных и задач.



5. Постановка задачи обучения на примерах

Описание объектов и ответов. Типы задач машинного обучения.

Обучение с учителем, предсказательные модели.

Приведите не менее 4-х примеров описания прикладных задач.

Постановка задачи обучения

В машинном обучении часто изучается неизвестная функция $\varphi(x)$, которая отображает **объекты** (входные данные) во **множество ответов**.

Объекты (X) — это элементы, описывающие ситуацию или пример (например, письмо, пациент, квартира).

Ответы (Y) — это результат, связанный с объектом (например, класс, цена, диагноз).

Имеется обучающая выборка:

$$\{(x_1, y_1), (x_2, y_2), \dots, (x_\ell, y_\ell)\}$$

где каждый $x_i \in X$, а $y_i = \varphi(x_i)$ — известный правильный ответ.

Задача: построить **модель** $g(x, \theta)$, приближающую $\varphi(x)$, то есть такую, что $g(x, \theta) \approx \varphi(x)$.

$\theta \in \Theta$ — вектор параметров модели.

Типы задач машинного обучения

1. Классификация

Ответ Y — дискретное множество меток классов.

Пример: определение, является ли письмо спамом.

2. Регрессия

Ответ $Y \subset \mathbb{R}$ — вещественное значение.

Пример: предсказание стоимости квартиры.

3. Ранжирование

Требуется упорядочить объекты по степени релевантности.

Пример: сортировка результатов поиска.

Кластеризация (*вне рамок обучения с учителем, но часто упоминается*)

Пример: группировка клиентов по поведению.

◆ **Обучение с учителем и предсказательные модели**

Обучение с учителем — это подход, при котором модель обучается на размеченных данных, где каждому объекту x_i соответствует правильный ответ y_i .

Предсказательная модель — это функция $g(x, \theta)$, способная выдавать предсказание y по новому входу x .

Цель: обобщить знания, полученные из обучающей выборки, и минимизировать ошибку предсказания на новых данных.

Примеры прикладных задач (не менее 4)

№	Прикладная задача	Объект (вход)	Ответ (выход)	Тип задачи
1	Классификация спама	Текст письма и мета-информация	{Спам, Не спам}	Классификация
2	Оценка стоимости недвижимости	Площадь, район, этаж	Цена в долларах	Регрессия
3	Диагностика заболеваний	Данные пациента	Тип болезни	Классификация
4	Кредитный скоринг	Доход, возраст, кредитная история	Риск невозврата (0–1)	Регрессия/Классификация
5	Рекомендация фильмов	История просмотров пользователя	Ранжированный список фильмов	Ранжирование

6. Описание объектов и ответов. Типы задач машинного обучения

❖ Описание объектов — Векторы признаков

Каждый **объект** в задаче машинного обучения описывается с помощью **вектора признаков**.

Обозначим:

- $f_j : X \rightarrow D_j$, где $j = 1, \dots, n$

Здесь:

- f_j — функция, извлекающая j -й признак объекта x
- D_j — множество значений признака

◆ Типы скалярных признаков:

1. Бинарный признак:

- $D_j = \{0, 1\}$
- Пример: есть_фото = 1, если пользователь загрузил фото, иначе 0

2. Номинальный (категориальный) признак:

- $|D_j| < \infty$, без естественного порядка
- Пример: цвет = {красный, синий, зелёный}

3. Порядковый признак:

- $|D_j| < \infty$, но значения упорядочены
- Пример: уровень удовлетворённости = {низкий < средний < высокий}

4. Качественный (числовой) признак:

- $D_j = \mathbb{R}$
- Пример: возраст, доход, температура

◆ Вектор признаков:

Для объекта x вектор признаков выглядит так:

$$(f_1(x), f_2(x), \dots, f_n(x)) \in \mathbb{R}^n$$

Это численное представление объекта, используемое алгоритмами.

◆ **Матрица признаков (дизайн-матрица):**

Если у нас ℓ объектов, то полная матрица признаков:

$$F = \begin{pmatrix} f_1(x_1) & \cdots & f_n(x_1) \\ \vdots & \ddots & \vdots \\ f_1(x_\ell) & \cdots & f_n(x_\ell) \end{pmatrix}_{\ell \times n}$$

Каждая строка — это объект, каждый столбец — это признак.

❖ **Описание ответов — Типы задач машинного обучения**

◆ **Задачи классификации**

Здесь необходимо отнести объект к одному или нескольким классам.

1. **Бинарная классификация:**

- $Y = \{-1, +1\}$ или $Y = \{0, 1\}$
- Пример: является ли письмо спамом?

2. **Многоклассовая классификация (взаимоисключающие классы):**

- $Y = \{1, \dots, M\}$
- Объект принадлежит **только одному классу**
- Пример: определение типа фрукта: {яблоко, банан, апельсин}

3. **Многоярлыковая (multilabel) классификация:**

- $Y = \{0, 1\}^M$
- Объект может принадлежать **нескольким классам одновременно**
- Пример: статья может быть отнесена к темам: {политика, экономика, спорт}

◆ **Задачи регрессии**

Здесь ответ Y — **вещественное значение**:

- $Y = \mathbb{R}$: одно числовое значение
 - Пример: предсказание цены квартиры
- $Y = \mathbb{R}^m$: несколько числовых значений
 - Пример: предсказание координат (x, y, z) объекта

◆ Задачи ранжирования

Модель должна выдать **упорядоченное множество** объектов:

- Y — конечное отсортированное множество
- Пример: ранжирование результатов поиска, рекомендации топ-5 товаров

✓ Сводная таблица:

Тип задачи	Целевое множество Y	Пример
Бинарная классификация	$\{0, 1\}$ или $\{-1, +1\}$	Определение наличия кошки на изображении
Многоклассовая классификация	$\{1, 2, \dots, M\}$	Распознавание цифр (0–9)
Многоярлыковая классификация	$\{0, 1\}^M$	Жанры фильма: {комедия, драма, боевик}
Регрессия	\mathbb{R}, \mathbb{R}^m	Предсказание стоимости недвижимости
Ранжирование	Упорядоченное множество	Сортировка кандидатов по

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

◆ Что такое обучение с учителем?

Обучение с учителем — это подход в машинном обучении, при котором модель обучается на **размеченных данных**, то есть на примерах, где каждому входу соответствует известный правильный ответ.

Этому процессу соответствуют такие понятия:

- Статистическое обучение с учителем
- Обучение по прецедентам
- Восстановление зависимости по эмпирическим данным
- Предсказательное моделирование
- Аппроксимация функции по заданным точкам

Все они описывают одну и ту же суть:

Построить функцию, приближающую истинную зависимость между входами и выходами по набору примеров.

◆ Цель предсказательной модели

Имея обучающую выборку:

$$\{(x_1, y_1), (x_2, y_2), \dots, (x_\ell, y_\ell)\}$$

нужно найти модель $g(x, \theta)$, которая бы для новых объектов давала ответы $g(x, \theta) \approx \varphi(x)$, где φ — неизвестная истинная функция.

Такая модель называется **предсказательной моделью**.

◆ Два основных типа задач:

1. Классификация

- Предсказание дискретной метки/класса
- Пример: классификация писем на спам и не-спам

2. Регрессия

- Предсказание вещественного числа
- Пример: предсказание цены недвижимости

◆ Не менее 4-х примеров прикладных задач

№	Задача	Объект (вход)	Ответ (выход)	Тип задачи	Источник
1	Обнаружение спама в письмах	Текст письма	{Спам, Не спам}	Классификация	Scikit-learn
2	Оценка стоимости недвижимости	Площадь, район, количество комнат	Цена в долларах	Регрессия	Scikit-learn
3	Кредитный скоринг	Кредитная история, доход, возраст	Уровень риска	Классификация / Регрессия	Scikit-learn
4	Медицинская диагностика	Симптомы, результаты анализов	Диагноз (например, грипп, COVID)	Классификация	Scikit-learn
5	Анализ отзывов	Текст отзыва	Положительный / Отрицательный	Классификация	Scikit-learn
6	Прогнозирование оценок студентов	Посещаемость, GPA, часы подготовки	Балл на экзамене (0–100)	Регрессия	Scikit-learn

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

◆ **Что такое алгоритм обучения?**

В задаче **обучения с учителем** весь процесс делится на два основных этапа:

1. Обучение (train)

На этом этапе **алгоритм обучения** получает размеченную выборку:

$$\mu : (X \times Y)^\ell \rightarrow \Theta$$

Это означает:

- $X \times Y$ — множество пар "объект-ответ"
- ℓ — число обучающих примеров
- μ — **алгоритм**, который по этим данным находит параметры модели $\theta \in \Theta$

По обучающей выборке:

$$X^\ell = \{(x_1, y_1), (x_2, y_2), \dots, (x_\ell, y_\ell)\}$$

алгоритм строит модель $g(x, \theta)$, **оценивая** параметры θ (обычно через минимизацию ошибки).

Таким образом, задача обучения **сводится к задаче оптимизации**.

2. Применение (test)

После обучения модель $g(x, \theta)$ используется для **предсказаний** на новых объектах x'_i .

На выходе получаем:

$$g(x'_i, \theta)$$

Эти предсказания можно сравнивать с реальными значениями (если они известны) для оценки качества модели.

📌 Итог: обучение = оптимизация

- Цель обучения — найти наилучшие параметры θ
- Это достигается **минимизацией функции ошибки**
- Следовательно, обучение сводится к **математической задаче оптимизации**

Обучение с учителем как оптимизация — Пошагово (Без кода)

Цель:

Построить модель $g(x, \theta)$, которая приближает неизвестную целевую функцию $\varphi(x)$

Шаг 1: Сбор обучающих данных

Имеется обучающая выборка — набор пар "объект — ответ":

$$(x_1, y_1), (x_2, y_2), \dots, (x_\ell, y_\ell)$$

Каждый x_i — это объект (вектор признаков), а y_i — правильный ответ (класс или значение).

Шаг 2: Выбор семейства моделей

Нужно выбрать тип модели, которую вы будете обучать. Например:

- Линейная модель: $g(x, \theta) = \theta_0 + \theta_1 x_1 + \dots + \theta_n x_n$
- Нейронная сеть, решающее дерево и др.

На этом этапе модель — это **функция с неизвестными параметрами θ** .

Шаг 3: Определение функции потерь

Функция потерь показывает, насколько плохо модель справляется с предсказаниями на обучающих данных.

Примеры:

- Для регрессии: $L(g(x_i, \theta), y_i) = (g(x_i, \theta) - y_i)^2$
- Для классификации: log-loss, кросс-энтропия и др.

Общая ошибка по всей выборке:

$$Q(\theta) = \sum_{i=1}^{\ell} L(g(x_i, \theta), y_i)$$

Шаг 4: Оптимизация — обучение модели

Теперь задача — **найти такие параметры θ** , при которых функция потерь **минимальна**:

$$\theta^* = \arg \min_{\theta} Q(\theta)$$

Это и есть **основная суть обучения** — решение задачи **математической оптимизации**.

Шаг 5: Модель обучена

Когда найдены оптимальные параметры θ^* , получается готовая модель:

$$g(x, \theta^*) \rightarrow \text{это и есть обученная модель}$$

Шаг 6: Применение модели (тестирование)

Теперь для любого **нового объекта x'** , модель выдает предсказание:

$$g(x', \theta^*) = \hat{y}$$

Это предсказание может быть классом, числом или рейтингом — в зависимости от типа задачи.

Образная метафора (без картинок):

- **Данные:** Точки на плоскости (или в пространстве)
- **Модель:** Кривая или поверхность, которую нужно провести через эти точки
- **Обучение:** Подбор формы кривой (изменение θ), чтобы она как можно лучше "обняла" данные
- **Предсказание:** Использование этой кривой для оценки новых, ещё не встречавшихся точек

9. Оценка качества. Эмпирический риск. Функция потерь

◆ Что такое функция потерь?

Функция потерь $\mathcal{L}(g, x)$ измеряет, насколько сильно ошибается алгоритм $g \in A$ на конкретном объекте $x \in X$.

Чем больше отличается предсказание $g(x, \theta)$ от правильного ответа $\varphi(x)$, тем выше значение функции потерь.

Иначе говоря:

| Функция потерь = штраф за ошибку на одном примере

◆ Функции потерь в разных типах задач

★ Для задач классификации:

$$\mathcal{L}(g, x) = [g(x, \theta) \neq \varphi(x)]$$

Это индикатор ошибки:

- 1 — если ответ **неверный**
- 0 — если **верный**

● Простая и наглядная метрика — часто называется **0-1 loss**

★ Для задач регрессии:

1. Абсолютная ошибка:

$$\mathcal{L}(g, x) = |g(x, \theta) - \varphi(x)|$$

- Показывает, насколько сильно предсказание отклоняется от истинного значения
- Линейное наказание за ошибку

2. Квадратичная ошибка:

$$\mathcal{L}(g, x) = (g(x, \theta) - \varphi(x))^2$$

- Ошибки больших размеров наказываются **сильнее**
- Широко используется в линейной регрессии

◆ Что такое эмпирический риск?

Эмпирический риск (иначе: средняя ошибка, training loss) — это **среднее значение функции потерь** по всей обучающей выборке:

$$R_{\text{emp}}(\theta) = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathcal{L}(g(x_i, \theta), y_i)$$

Где:

- ℓ — количество примеров
- $g(x_i, \theta)$ — предсказание
- $y_i = \varphi(x_i)$ — истинный ответ

🎯 Цель обучения:

Найти параметры θ , минимизирующие эмпирический риск — это и есть задача оптимизации.

10. Что такое переобучение и недообучение? Как их избежать?

◆ Недообучение (Underfitting)

Определение:

Недообучение возникает, когда модель **слишком простая** и не способна уловить зависимости в данных.

⭐ Признаки:

- Много данных
- Недостаточно параметров
- Модель **жесткая**, негибкая
- Высокая ошибка как на обучении, так и на teste

💬 Пример:

Пытаемся аппроксимировать нелинейные данные **прямой линией** → модель не улавливает закономерности.

◆ Переобучение (Overfitting)

Определение:

Переобучение возникает, когда модель **слишком сложная** и начинает "запоминать" не только закономерности, но и **шум** в обучающих данных.

❖ **Признаки:**

- Мало данных
- Слишком много параметров
- Модель **гипергибкая**, "запоминает" данные
- Низкая ошибка на обучении
- Высокая ошибка на teste

👉 **Пример:**

Полином высокой степени проходит через все обучающие точки, но дает нелогичные предсказания на новых данных.

📈 Графическое представление:

- **Недообучение** — модель не отражает закономерности → плохо на обучении и teste
- **Хорошая модель** — оптимальная сложность → хорошая обобщающая способность
- **Переобучение** — модель подгоняется под шум → хорошо на обучении, плохо на teste

Также изображается кривая зависимости **качества модели (Q)** от **сложности**:

- Качество сначала растет с увеличением сложности
- Достигает пика (оптимум)
- Затем падает из-за переобучения

🔍 **Почему возникает переобучение?**

- Слишком много параметров у модели $g(x, \theta)$
- Параметры «расходуются» на подгонку к шуму в обучающей выборке
- Модель выбирается из сложного множества A , но на основе ограниченного количества данных X_ℓ



Как обнаружить переобучение?

- Эмпирически: разделить данные на обучающую и тестовую выборки
- Если:
 - Ошибка на обучении мала
 - А ошибка на teste велика
 - это признак переобучения



Как минимизировать переобучение?

1. Увеличить объем обучающих данных
 - Больше данных — лучше обобщение
2. Регуляризация
 - Наложить штрафы на большие веса параметров (например, L1/L2)
 - Это "сжимает" модель
3. Выбор модели по обобщающей способности
 - Использовать кросс-валидацию
 - Оценивать модель на неиспользованных данных
4. Минимизация теоретических оценок ошибки
 - Например, по VC-мере или структурному риску

11. Одномерная и многомерная линейная регрессия

◆ Одномерная линейная регрессия

- **Вход:** один признак $x \in \mathbb{R}$
- **Выход:** одно число $y \in \mathbb{R}$
- Модель: обычная прямая линия

$$g(x, \theta) = \theta_0 + \theta_1 x$$

- **Цель:** найти наилучшую прямую, аппроксимирующую данные
- **Метод:** минимизация суммы квадратов ошибок

$$Q(\theta) = \sum_{i=1}^{\ell} (\theta_0 + \theta_1 x_i - y_i)^2 \rightarrow \min_{\theta}$$

◆ Многомерная линейная регрессия

Если у объекта несколько признаков:

- $x_i = (x_{i1}, x_{i2}, \dots, x_{in}) \in \mathbb{R}^n$
- $y_i \in \mathbb{R}$ — скалярный отклик
- Целевая функция $\varphi : X \rightarrow Y$, которую мы приближаем моделью $g(x, \theta)$

◆ Форма модели:

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

- $f_j(x)$ — j-й числовый признак
- θ_j — параметр модели, отвечающий за значимость признака

◆ **Обучающая выборка:**

$$X^\ell = \{(x_1, y_1), (x_2, y_2), \dots, (x_\ell, y_\ell)\}$$

◆ **Функция ошибки (по МНК):**

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

Цель — найти такие параметры θ , при которых эта сумма минимальна.

◆ **Матричная форма записи:**

Чтобы упростить вычисления, используем матрицы:

Матрица признаков:

$$F_{\ell \times n} = \begin{pmatrix} f_1(x_1) & \cdots & f_n(x_1) \\ \vdots & \ddots & \vdots \\ f_1(x_\ell) & \cdots & f_n(x_\ell) \end{pmatrix}$$

Вектор ответов:

$$y_{\ell \times 1} = \begin{pmatrix} y_1 \\ \vdots \\ y_\ell \end{pmatrix}$$

Вектор параметров:

$$\theta_{n \times 1} = \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_n \end{pmatrix}$$

Финальный функционал:

$$Q(\theta) = \|F\theta - y\|^2 \rightarrow \min_{\theta}$$

Это и есть суммарная квадратичная ошибка — минимизация которой даёт наилучшую линейную модель.

12. Конструирование признаков

Что такое конструирование признаков?

Конструирование признаков (Feature Engineering) — это процесс создания новых признаков на основе исходных данных, используя интуицию, преобразования или прикладные знания.

Цель: улучшить качество модели за счёт более содержательных и полезных входных данных.

◆ Пример: прогнозирование стоимости квартиры

Исходные признаки:

- x_1 : площадь квартиры в м²
- x_2 : район или город (категориальный признак)

Начальная модель:

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

Предполагается, что влияние площади и района **аддитивное**.

Улучшение:

Вводим новый признак:

$$x_3 = x_1 \cdot x_2$$

Он напрямую отражает **стоимость квадратного метра в разных районах**.

Обновлённая модель:

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

Теперь модель учитывает, что **влияние площади зависит от района**.

Способы конструирования признаков:

1. Полиномиальные признаки

- Возведение признаков в степень, перемножение признаков

- Пример: x^2 , $x_1 \cdot x_2$

2. Агрегации

- Среднее, сумма, медиана и др. (особенно полезно для временных рядов и группировок)

Пример: средний доход в регионе

3. Лаги (временные сдвиги)

- Значения признаков в прошлые моменты времени

Пример: продажи за прошлую неделю

4. Темпоральные признаки

- День недели, месяц, номер квартала
- Учитывают сезонность

5. Экспертные знания

Признаки, созданные на основе понимания предметной области

Пример: отношение мощности двигателя к весу автомобиля

13. Нормализация признаков

Что такое нормализация признаков?

Нормализация — это преобразование значений признаков к единой шкале, чтобы:

- Повысить качество обучения
- Сделать все признаки сравнимыми по масштабу

Это особенно важно при использовании градиентных методов, SVM и линейных моделей.

◆ 1. Z-нормализация (нормализация средним)

Также называется **стандартизация**:

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

Где:

- μ_{x_j} — среднее значение признака
- σ_{x_j} — стандартное отклонение

★ Результат:

- Среднее = 0
- Стандартное отклонение = 1

Используется, если важна форма распределения и масштаб.

◆ 2. Минимаксная нормализация

Приведение значений признака к диапазону, например, [0, 1] или [-1, 1]:

$$x_j^* = \frac{x_j - \min(x_j)}{\max(x_j) - \min(x_j)}$$

Можно использовать также **среднее** или **медиану** вместо минимума.

Когда нужна нормализация?

Нормализация обязательна, если:

- Признаки имеют разные масштабы
- Один признак может "заглушать" другие из-за больших чисел
- Модель чувствительна к масштабу признаков (SVM, градиентный спуск)

Эвристика:

Постарайтесь привести все признаки к диапазону:

$$-1 \leq x_j \leq 1$$

Если диапазон признака **отличается от** $[-1, 1]$ более чем на **2 порядка**, лучше нормализовать.

◆ Примеры:

Нормализация не требуется:

- $-3 \leq x_1 \leq 3$
- $-0.3 \leq x_2 \leq 0.3$
- $0 \leq x_3 \leq 3$
- $-2 \leq x_4 \leq 0.5$

Нормализация обязательна:

- $-100 \leq x_5 \leq 100$
- $-0.001 \leq x_6 \leq 0.001$
- $98.6 \leq x_7 \leq 105$

Лучше выполнить нормализацию признаков, чем её не делать — особенно при большом разбросе значений.

14. Метод наименьших квадратов (МНК)

Что это такое?

Метод наименьших квадратов (МНК) — это классический способ обучения моделей (особенно линейных), при котором параметры подбираются так, чтобы сумма квадратов ошибок между предсказанными и фактическими значениями была минимальной.

Используется для нахождения такой модели, которая как можно лучше приближает наблюдаемые данные.

◆ Формальная постановка задачи

Пусть у нас есть обучающая выборка:

$$X^\ell = \{(x_1, y_1), (x_2, y_2), \dots, (x_\ell, y_\ell)\}$$

Где:

- $x_i \in \mathbb{R}^n$ — вектор признаков
- $y_i \in \mathbb{R}$ — реальный ответ
- $g(x, \theta)$ — модель с параметрами θ

◆ Целевая функция МНК (функционал ошибки):

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

◆ Пример: линейная регрессия

Модель:

$$g(x, \theta) = \theta_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

В матричной форме:

- F — матрица признаков (размер $\ell \times n$)
- y — вектор ответов
- θ — вектор параметров

Тогда задача:

$$Q(\theta) = \|F\theta - y\|^2 \rightarrow \min_{\theta}$$

◆ Решение (нормальное уравнение):

Условие минимума:

$$\frac{\partial Q}{\partial \theta} = 2F^T(F\theta - y) = 0$$

Получаем нормальную систему уравнений:

$$F^T F\theta = F^T y$$

Решение:

$$\theta^* = (F^T F)^{-1} F^T y$$

⚠ При условии, что $F^T F$ — обратима (иначе используют регуляризацию).

◆ Зачем нужен МНК?

- Прост в реализации
- Обеспечивает **оптимальную аппроксимацию** в смысле среднеквадратичной ошибки
- Основа для более сложных методов (ридж-регрессия, LASSO, линейная классификация)

📈 Интерпретация:

Метод наименьших квадратов подбирает такую "линию", "плоскость" или гиперплоскость, которая **минимизирует общее отклонение точек** данных от этой поверхности (в смысле квадратов вертикальных отклонений).

✓ Вывод:

МНК — базовый и мощный инструмент для регрессионного анализа и построения предсказательных моделей.



15. Алгоритм градиентного спуска (Gradient Descent Algorithm)

Что это такое?

Градиентный спуск — это численный метод минимизации функции (например, функции потерь), при котором параметры модели итеративно обновляются в направлении антиградиента (то есть туда, где функция убывает быстрее всего).

-Он используется для нахождения оптимальных параметров модели θ , минимизирующих ошибку обучения.

◆ Общий вид итерации:

$$\theta_j^{\text{next}} = \theta_j^{\text{prev}} - \alpha \cdot \frac{\partial Q(\theta)}{\partial \theta_j}$$

- α — шаг обучения (learning rate)
- $Q(\theta)$ — функция ошибки
- $\frac{\partial Q}{\partial \theta_j}$ — частная производная функции по параметру θ_j

◆ Критерии остановки (когда завершать обучение):

Итерации продолжаются, пока не выполнится одно из условий:

1. Параметры почти не меняются:

$$|\theta_j^{\text{next}} - \theta_j^{\text{prev}}| < \delta_\theta \quad \text{для всех } j$$

2. Функция ошибки почти не уменьшается:

$$|Q(\theta^{\text{prev}}) - Q(\theta^{\text{next}})| < \delta_Q$$

Важно!

Все параметры θ_j должны обновляться одновременно, а не последовательно.

Как влияет шаг обучения α ?

-Слишком маленький шаг: обучение будет медленным, много итераций

-Слишком большой шаг: возможна расходимость, модель будет прыгать мимо минимума

◆ Пример: линейная регрессия с одним признаком

Модель:

$$g(x_i, \theta_0, \theta_1) = \theta_1 x_i + \theta_0$$

- ◆ **Функция ошибки (среднеквадратичная):**

$$Q(\theta_0, \theta_1) = \frac{1}{2\ell} \sum_{i=1}^{\ell} (\theta_1 x_i + \theta_0 - y_i)^2$$

- ◆ **Частные производные:**

$$\frac{\partial Q}{\partial \theta_0} = \frac{1}{\ell} \sum_{i=1}^{\ell} (\theta_1 x_i + \theta_0 - y_i)$$

$$\frac{\partial Q}{\partial \theta_1} = \frac{1}{\ell} \sum_{i=1}^{\ell} (\theta_1 x_i + \theta_0 - y_i) \cdot x_i$$

- ◆ **Финальные формулы обновления параметров:**

$$\theta_0 := \theta_0 - \alpha \cdot \frac{1}{\ell} \sum_{i=1}^{\ell} (g(x_i, \theta) - y_i)$$

$$\theta_1 := \theta_1 - \alpha \cdot \frac{1}{\ell} \sum_{i=1}^{\ell} (g(x_i, \theta) - y_i) \cdot x_i$$

- **Что происходит на каждой итерации:**

1. Считаем ошибку модели на всех объектах
2. Считаем градиенты (производные ошибки по параметрам)
3. Двигаемся в сторону уменьшения ошибки
4. Повторяем до достижения минимума

Градиентный спуск — это ключевой алгоритм оптимизации, позволяющий обучать модели, когда прямое решение невозможно или неэффективно.

16. Стохастический градиентный спуск (SGD — Stochastic Gradient Descent)

Проблема с обычным градиентным спуском:

Когда обучающая выборка очень большая ($\ell \gg 0$), классический градиентный спуск (GD) становится медленным, потому что:

- На каждой итерации нужно пересчитать сумму градиентов по всей выборке
- Вычислительная нагрузка растёт пропорционально объёму данных

Решение: Стохастический градиентный спуск (SGD)

SGD обновляет параметры после каждого отдельного примера, а не после всей выборки.

➤ Вместо того, чтобы минимизировать функционал качества $Q(\theta)$, мы минимизируем функцию потерь на одном объекте:

$$L_i(g, x_i) = (g(x_i, \theta) - y_i)^2$$

◆ Алгоритм стохастического градиентного спуска:

1. Выбрать случайный пример из обучающей выборки:

$$(x_i, y_i) \in X^\ell$$

2. Вычислить ошибку на этом примере:

$$\varepsilon_i = L_i(g, x_i) = (g(x_i, \theta) - y_i)^2$$

3. Обновить параметры θ по градиенту только этого примера:

$$\theta_j := \theta_j - \alpha \cdot \frac{\partial}{\partial \theta_j} L_i(g, x_i)$$

Где:

- α — шаг обучения (learning rate)
- Обновление выполняется **мгновенно** после одного примера

4. Обновить приближённую оценку функции качества (сглаженный Q):

$$\bar{Q} := \lambda \cdot \varepsilon_i + (1 - \lambda) \cdot \bar{Q}$$

Где:

- $\lambda \in [0, 1]$ — скорость забывания (чем больше, тем быстрее "забываются" старые ошибки)

5. Повторять шаги 1–4 до:

- Сходимости параметров θ
- Или стабилизации \bar{Q}

Сравнение: GD vs. SGD

Характеристика	Градиентный спуск (GD)	Стохастический градиентный спуск (SGD)
Скорость обновления	После всей выборки	После одного примера
Вычисления на итерацию	Дорогие (вся выборка)	Лёгкие и быстрые
Шум	Почти нет	Высокий, но помогает избежать минимумов
Сходимость	Плавная, медленная	Быстрая, но "скачет"

Стохастический градиентный спуск — это быстрая и эффективная альтернатива обычному градиентному методу, особенно на больших данных. Стохастический градиентный спуск — это быстрая и эффективная альтернатива обычному градиентному методу, особенно на больших данных.

17. Инициализация весов и шаг обучения в градиентном спуске

Зачем это важно: начальные значения весов θ и шаг α сильно влияют на скорость сходимости и качество решения. Плохая инициализация может привести к медленному обучению, расходимости или плохому минимуму.

◆ Варианты инициализации весов θ :

1. Нули: $\theta_j = 0$. Просто, но неэффективно в сложных моделях (особенно в нейросетях — приводит к симметрии).
2. Малые случайные значения: $\theta_j = \text{random}\left(-\frac{1}{2n}, \frac{1}{2n}\right)$. Часто используется, помогает избежать симметрии, улучшает сходимость.
3. Корреляционная оценка:

$$\theta_j = \frac{\langle y, f_j \rangle}{\langle f_j, f_j \rangle}, \quad f_j = (f_j(x_i))_{i=1}^\ell$$

Оптимально при квадратичной функции потерь и некоррелированных признаках.

4. Подвыборка: обучение на случайной малой части выборки → использовать результат как стартовую точку. Особенно полезно при больших данных.

5. Мультистарт: несколько запусков с разных стартов, выбор лучшего по значению ошибки. Увеличивает шанс найти глобальный минимум.

◆ **Шаг обучения α :**

Маленький шаг: медленно, требуется много итераций.

Большой шаг: возможно расходимость, прыжки мимо минимума.

Как выбирать:

- Фиксированный (например, 0.01)
- Затухающие: $\alpha_k = \frac{\alpha_0}{1+\lambda k}$
- Адаптивные: методы Adam, RMSprop, Adagrad подбирают шаг автоматически.

Вывод: правильная инициализация и шаг обучения — ключ к эффективному и стабильному обучению модели.

18. Регуляризация в градиентном спуске для борьбы с переобучением

Цель: уменьшить переобучение путём добавления штрафа за слишком большие значения весов θ . Модель с большими весами более гибкая и склонна к переобучению.

◆ **Модифицированная функция потерь:**

Добавляется регуляризационный член — штраф за норму весов:

$$\tilde{L}_i(\theta) = L_i(\theta) + \frac{\tau}{2} \|\theta\|^2 = L_i(\theta) + \frac{\tau}{2} \sum_{j=1}^n \theta_j^2$$

Здесь:

- $L_i(\theta)$ — обычная функция потерь на одном примере
- τ — коэффициент регуляризации (чем больше, тем сильнее "сжимает" веса)

◆ Градиент с регуляризацией:

$$\nabla \tilde{L}_i(\theta) = \nabla L_i(\theta) + \tau\theta$$

То есть обычный градиент плюс дополнение, тянущее веса к нулю.

◆ Модифицированный шаг градиентного спуска:

$$\theta = \theta(1 - \alpha\tau) - \alpha\nabla L_i(\theta)$$

- Первая часть: "усушка" весов
- Вторая часть: стандартный градиентный шаг

Регуляризация замедляет рост весов и помогает лучше обобщать на тестовых данных.

◆ Подбор коэффициента регуляризации τ :

1. Скользящий контроль (hold-out validation): делим данные на train/val, подбираем τ , дающий минимальную ошибку на валидации.
2. Стохастическая адаптация: подбираем τ динамически во время обучения, на основе поведения ошибки.
3. Байесовский подход (двухуровневый вывод): рассматриваем τ как параметр вероятностной модели, подбираем его через аппроксимацию апостериорного распределения.

Вывод: регуляризация — простой и мощный способ снизить переобучение, добавляя штраф к функции ошибки. Это делает модель более устойчивой и обобщающей.

19. Повышение производительности с помощью векторизации

Идея: векторизация — это переписывание вычислений так, чтобы вместо поэлементных циклов использовать матричные/векторные операции. Это **резко ускоряет обучение**, особенно на больших данных.

Пример: регрессионная модель

Скалярно (медленно):

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

где $x_{i,0} = 1$ — фиктивный признак (bias).

В коде:

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

Векторно (быстро):

$$g(x_i, \theta) = \theta \cdot x_i$$

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

Почему быстрее:

- Используются оптимизированные библиотеки (NumPy, BLAS)
- Векторные инструкции на уровне CPU (SIMD)
- Убираются циклы Python — ускорение в 10–100 раз

Расширение на матрицы:

Если F — матрица признаков всех объектов, и θ — вектор весов, то предсказания сразу для всех:

$$\hat{y} = F \cdot \theta$$

```
python  
  
y_pred = np.dot(F, theta)
```

Векторизация — это ключ к высокой производительности в машинном обучении. Она ускоряет вычисления, делает код компактнее и эффективнее. Используется повсеместно в реальных ML-системах.

20. Постановка задачи бинарной и многоклассовой классификации

Бинарная классификация:

Обучающая выборка:

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

Модель: линейная классификация

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

Функция потерь:

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

Метод обучения — минимизация эмпирического риска:

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

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:

Обучающая выборка:

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

Модель: для каждого класса $y \in Y$ — отдельный вектор весов θ_y

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

Функция потерь:

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

Метод обучения — минимизация эмпирического риска:

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

Проверка на тестовой выборке $X^k = \{(\tilde{x}_i, \tilde{y}_i)\}_{i=1}^k$ аналогична: предсказывается класс $\arg \max_y \langle \tilde{x}_i, \theta_y \rangle$, затем сравнивается с \tilde{y}_i .

Вывод: бинарная классификация — это частный случай многоклассовой, в которой число классов $|Y| = 2$, и модель делит пространство линейной границей; в многоклассовой задаче выбирается класс с максимальным "счётом". В обеих задачах минимизируется эмпирический риск на основе соответствующей функции потерь.

21. Классификаторы на основе отступа (margin-based) (на русском)

Бинарный классификатор:

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

$h(x, \theta)$ — дискриминантная функция, определяющая границу между классами.

Разделяющая поверхность — там, где $h(x, \theta) = 0$.

Отступ:

$$M_i(\theta) = h(x_i, \theta) \cdot y_i$$

Отступ показывает, насколько уверенно классифицируется объект x_i .

Если $M_i(\theta) < 0$, значит классификатор ошибается на x_i .

Ранжирование объектов по отступу:

- Малый отступ: шумовые точки
- Отступ ≈ 0 : пограничные
- Большой отступ: уверенно классифицированные (надёжные)

Многоклассовый классификатор:

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

Каждому классу $y \in Y$ соответствует своя дискриминантная функция h_y .

Поверхность $h_y = h_z$ разделяет классы y и z .

Отступ объекта x_i от неверного класса y :

$$M_i^y(\theta) = h_{y_i}(x_i, \theta_{y_i}) - h_y(x_i, \theta_y)$$

Итоговый (наименьший) отступ:

$$M_i(\theta) = \min_{y \neq y_i} M_i^y(\theta)$$

Если $M_i(\theta) < 0$, классификатор ошибается на x_i .

Вывод: Классификаторы с отступом (например, SVM) учитывают не только метку класса, но и **уверенность** в решении. Чем больше отступ, тем надёжнее классификация.

Двухклассовая (бинарная) логистическая регрессия

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

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

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

Логарифмическая функция потерь:

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

Модель условной вероятности:

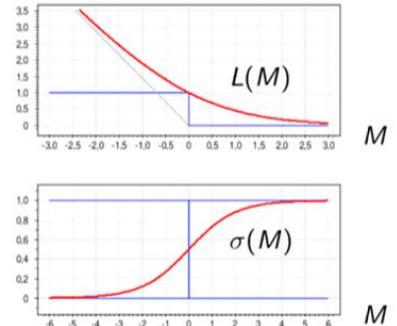
$$P(y = 1|x; \theta) = \sigma(M) = \frac{1}{1 + e^{-M}}$$

где $\sigma(M)$ – сигмоидная (логистическая) функция,

важное свойство: $\sigma(M) + \sigma(-M) = 1$.

Максимизация правдоподобия (logistic loss) с регуляризацией:

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



Мноклассовая логистическая регрессия

Линейный классификатор при произвольном числе классов $|Y|$:

$$g(x) = \arg \max_{y \in Y} \langle \theta_y, x \rangle, \quad x, \theta_y \in \mathbb{R}^n$$

Вероятность того, что объект x относится к классу y_k :

$$P(y = y_k | x; \theta) = \frac{\exp(\langle \theta_{y_k}, x \rangle)}{\sum_{z \in Y} \exp(\langle \theta_z, x \rangle)} = \text{SoftMax}_{y \in Y} \langle \theta_y, x \rangle$$

функция SoftMax: $\mathbb{R}^Y \rightarrow \mathbb{R}^Y$ переводит произвольный вектор в нормированный вектор дискретного распределения.

Максимизация правдоподобия (log-loss) с регуляризацией:

$$Q_{MAP}(\theta) = \sum_{i=1}^{\ell} \log P(y_i | x_i, \theta) - \frac{\tau}{2} \sum_{y \in Y} \|\theta_y\|^2 \rightarrow \max_{\theta}$$

Здесь рассматривается бинарная логистическая регрессия, в которой линейная модель предсказывает не только метку класса, но и вероятность принадлежности объекту к классу $+1$. Для этого используется сигмоида, переводящая линейный ответ в диапазон от 0 до 1. Модель обучается путём минимизации логарифмической функции потерь, которая штрафует за ошибки. Для борьбы с переобучением добавляется регуляризация, уменьшающая значения весов.

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

Вероятностный подход. Принцип максимизации правдоподобия

- Пусть $X \times Y$ – вероятностное пространство с плотностью $p(x, y)$
- Пусть X^ℓ – простая (i.i.d., independent identically distributed) выборка: $(x_i, y_i)_{i=1}^\ell \sim p(x, y)$
- **Задача:** по выборке X^ℓ оценить плотность $p(x, y)$

$$p(x, y) = P(y|x; \theta)p(x) \text{ – параметризация плотности:}$$

$$P(y|x; \theta) \text{ – условная вероятность класса } y;$$

$$p(x) \text{ – неизвестное и непараметризуемое распределение на } X.$$

- Максимум правдоподобия (Maximum Likelihood Estimate, MLE):

$$p(X^\ell, \theta) = \prod_{i=1}^\ell p(x_i, y_i) = \prod_{i=1}^\ell P(y_i|x_i; \theta)p(x_i) \rightarrow \max_{\theta}$$

- Максимум логарифма правдоподобия (log-likelihood, log-loss):

$$Q_{MLE}(\theta) = \sum_{i=1}^\ell \log P(y_i|x_i, \theta) \rightarrow \max_{\theta}$$

В **математической статистике** метод максимального правдоподобия — это метод оценки параметров вероятностного распределения с помощью максимизации **функции правдоподобия**. Точка в пространстве параметров, которая максимизирует функцию правдоподобия, называется оценкой максимального правдоподобия.

Предположим, что имеется некоторый набор наблюдений, который является случайной **выборкой** из некоторой неизвестной **совокупности**. Целью оценки максимального правдоподобия являются выводы о совокупности, из которой была сформирована выборка, в частности, о совместном распределении вероятностей случайных величин (y_1, y_2, \dots, y_n) , не обязательно независимых и одинаково распределенных.

С каждым распределением вероятностей связан уникальный вектор $\theta = (\theta_1, \theta_2, \dots, \theta_k)^T$, которые индексируют распределение вероятностей в параметрическом семействе. Пространство параметров полагается евклидовым и имеющим конечную размерность.

Здесь рассматривается вероятностный подход к обучению моделей. Принцип максимального правдоподобия (MLE) заключается в выборе таких параметров модели, при которых наблюдаемые данные максимально вероятны. Так как распределение входов неизвестно, моделируется условная вероятность выхода при заданном входе. Для удобства вычислений берётся логарифм правдоподобия (логарифмическая функция), которую затем максимизируют. В задачах с учителем максимизация логарифмического правдоподобия эквивалентна минимизации эмпирического риска, если функция потерь построена на вероятностной модели (например, логистическая).

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

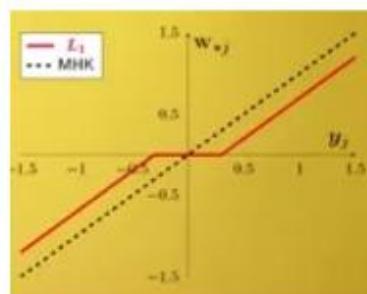
L1-регуляризация (англ. lasso regression), или регуляризация через манхэттенское расстояние:

$$L_1 = \sum_i (y_i - y(t_i))^2 + \lambda \sum_i |a_i|.$$

Лосс-функция с L1 регуляризацией (желтая рамка).

Если короче, то прибавление к лосс-функции суммы весов в модуле. Она добавляет «критическую величину» коэффициента, как дополнение к функции потерь. То есть, если весы на расстоянии лямбда ближе к нулю, то они становятся равными нулю и уходят в игнор, тем самым отбирая только важные признаки и в следствии уменьшая саму сеть, что экономит ресурсы.

В графике расстояние лямбда прижимается к нулю. Это расстояние и выключает низкие веса.



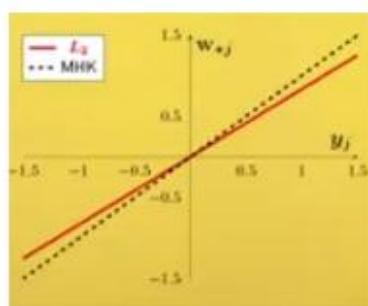
L2-регуляризация, или регуляризация Тихонова (в англоязычной литературе – ridge regression или Tikhonov regularization), для интегральных уравнений позволяет балансировать между соответствием данным и маленькой нормой решения:

$$L_2 = \sum_i (y_i - y(t_i))^2 + \lambda \sum_i a_i^2$$

Лосс-функция с L2 регуляризацией (желтая рамка).

Тут получается что функция занижает пики прибавляя сумму весов в квадрате с множителем лямбда.

В графике происходит легкое смещение (не дисперсия) добавляя статистическую ошибку, которая не влияет на точность, но позволяет не переобучиться. Соответственно если вы поставите большую коэффициент (лямбда) L2, то система никогда не обучится, просто из-за того что сильно сместиться. Но если лямбда будет маленькой то и смещение будет маленьким, и весь смысл регуляризации так же будет минимальным.



25. Понятие расстояния между объектами. Метрика Минковского.

- 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.

26. Обобщенный метрический классификатор. Метод k ближайших соседей.

Обобщенный метрический классификатор

- Для произвольного $x \in X$ отранжируем объекты x_1, \dots, x_ℓ :

$$\rho(x, x^{(1)}) \leq \rho(x, x^{(2)}) \leq \dots \leq \rho(x, x^{(\ell)})$$

$x^{(i)}$ – i -ый сосед объекта x среди x_1, \dots, x_ℓ ;

$y^{(i)}$ – ответ на i -м соседе объекта x .

- Метрический алгоритм классификации относит объект x к тому классу, которому принадлежат его ближайшие соседи:

$$g(x; X^\ell) = \arg \max_{y \in Y} \underbrace{\sum_{i=1}^{\ell} [y^{(i)} = y] w(i, x)}_{\Gamma_y(x)}$$

$w(i, x)$ – вес, степень близости к объекту x его i -го соседа, неотрицателен, не возрастает по i .

$\Gamma_y(x)$ – оценка близости объекта x к классу y .

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.

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.

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

Разделяющая полоса (разделяющая гиперплоскость посередине):

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

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

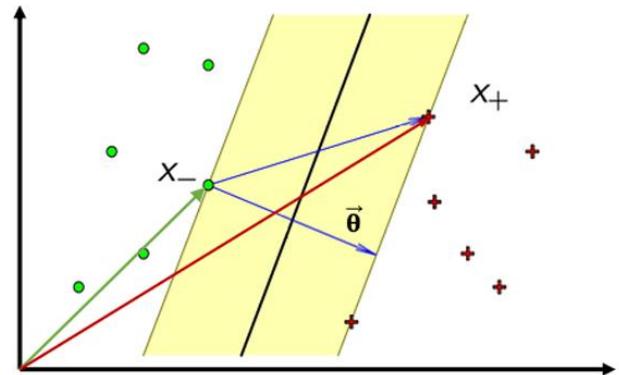
$$\exists \vec{x}_-: \langle \vec{x}_-, \vec{\theta} \rangle - \theta_0 = -1$$

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

Ширина полосы:

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

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



Справка: $\langle u, v \rangle = \|u\| \|v\| \cos \alpha, \nu$

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:

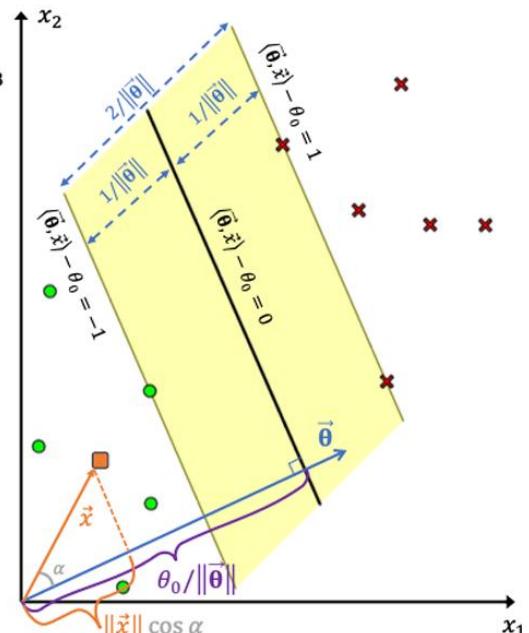
Чтобы узнать, по какую сторону от разделяющей гиперплоскости находится объект \vec{x} , достаточно спроектировать вектор \vec{x} на вектор $\vec{\theta}$ и сравнить длину получившейся проекции с пороговым значением θ_0 :

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

Решающее правило:

$$\langle \vec{\theta}, \vec{x} \rangle - \theta_0 \geq 0 \Rightarrow \vec{x} \in X_+$$

$$\langle \vec{\theta}, \vec{x} \rangle - \theta_0 < 0 \Rightarrow \vec{x} \in X_-$$



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

Метод ККТ – обобщение метода множителей Лагранжа

- Задача математического программирования:

$$\begin{cases} f(x) \rightarrow \min_x; \\ g_i(x) \leq 0, \quad i = 1, \dots, m; \\ h_j(x) = 0, \quad j = 1, \dots, k. \end{cases}$$

- Необходимые условия. Если x – точка локального минимума, то существуют множители Лагранжа $\mu_i, i = 1, \dots, m, \lambda_j, j = 1, \dots, k$:

$$\begin{cases} \frac{\partial L}{\partial x} = 0, \quad L(x; \mu, \lambda) = f(x) + \sum_{i=1}^m \mu_i g_i(x) + \sum_{j=1}^k \lambda_j h_j(x) \\ g_i(x) \leq 0; \quad h_j(x) = 0; \quad (\text{исходные ограничения}) \\ \mu_i \geq 0; \quad (\text{двойственные ограничения}) \\ \mu_i g_i(x) = 0; \quad (\text{условие дополняющей нежесткости}) \end{cases}$$

где $L(x; \mu, \lambda)$ – функция Лагранжа

30. Понятие опорного вектора и типизация объектов.

Понятие опорного вектора и типизация объектов

Система условий ККТ:

$$\begin{cases} \vec{\theta} = \sum_{i=1}^{\ell} \lambda_i y_i \vec{x}_i; \quad \sum_{i=1}^{\ell} \lambda_i y_i = 0; \quad M_i(\vec{\theta}, \theta_0) = 1 - \xi_i; \\ \xi_i \geq 0, \quad \lambda_i \geq 0, \quad \eta_i \geq 0, \quad \eta_i + \lambda_i = C; \\ \lambda_i = 0 \text{ либо } M_i(\vec{\theta}, \theta_0) = 1 - \xi_i; \\ \eta_i = 0 \text{ либо } \xi_i = 0; \end{cases}$$

Определение. Объект \vec{x}_i называется **опорным**, если $\lambda_i \neq 0$.

Типизация объектов $\vec{x}_i, i = 1, \dots, \ell$:

- $\lambda_i = 0; \eta_i = C; \xi_i = 0; M_i \geq 1$ – периферийный.
- $0 < \lambda_i < C; 0 < \eta_i < C; \xi_i = 0; M_i = 1$ – **опорный**-граничный.
- $\lambda_i = C; \eta_i = 0; \xi_i > 0; M_i < 1$ – **опорный**-нарушитель.

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

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

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

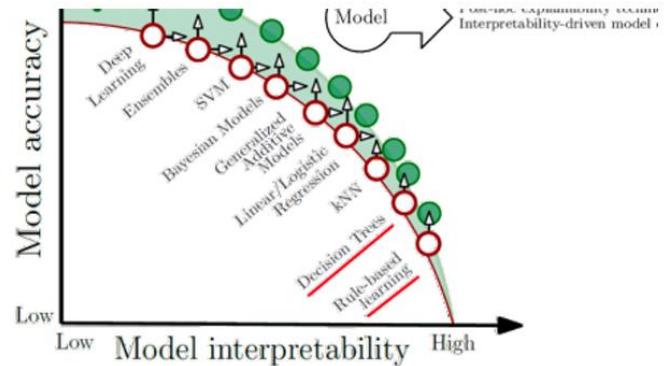
6. Сеть радиальных базисных функций (RBF ядро, гауссовское ядро)

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

32. Интерпретируемость алгоритмов машинного обучения.

XAI, eXplainable AI

- **Interpretability** – пассивная интерпретируемость внутреннего строения модели или предсказания на объекте
- **Understandability, Transparency** – понятность, самоочевидность, прозрачность строения модели
- **Explainability** – активная генерация объяснений на объекте как дополнительных выходных данных модели
- **Comprehensibility** – возможность представить выученные закономерности в виде понятного людям знания



“Do you want an interpretable model or the one that works?”

[Yann LeCun, NIPS’17]

NIPS 2017 AI Debate: <https://youtu.be/93Xv8vJ2acl>

33. Деревья принятия решений. Определение, алгоритмы построения.

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, $\forall v \in V_{\text{внутр}}$;

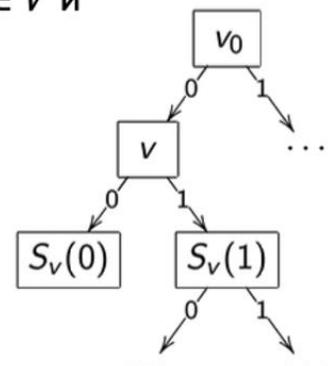
$S_v: D_v \rightarrow V$ – множество дочерних вершин;

$y_v \in Y$ – 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**

34. Критерий Джинни, энтропийный критерий.

Информативность в задаче классификации: критерий Джинни

Пусть предсказание модели — это распределение вероятностей классов (c_1, \dots, c_K) . Вместо логарифма правдоподобия в качестве критерия можно выбрать, например, [метрику Бриера](#) (за которой стоит всего лишь идея посчитать MSE от вероятностей). Тогда информативность получится равной

$$H(X_m) = \min_{\sum_k c_k = 1} \frac{1}{|X_m|} \sum_{(x_i, y_i) \in X_m} \sum_{k=1}^K (c_k - \mathbb{I}[y_i = k])^2$$

Можно показать, что оптимальное значение этой метрики, как и в случае энтропии, достигается на векторе c , состоящем из выборочных оценок частот классов (p_1, \dots, p_K) . $p_i = \frac{1}{|X_m|} \sum_i \mathbb{I}[y_i = k]$. Если подставить (p_1, \dots, p_K) в выражение выше и упростить его, получится [критерий Джинни](#):

$$H(X_m) = \sum_{k=1}^K p_k (1 - p_k)$$

Критерий Джинни допускает и следующую интерпретацию: $H(X_m)$ равно математическому ожиданию числа неправильно классифицированных объектов в случае, если мы будем приписывать им случайные метки из дискретного распределения, заданного вероятностями (p_1, \dots, p_K) .

▼ Немного подробнее об энтропии

Величина

$$H(X_m) = - \sum_k p_k \log p_k$$

называется [информационной энтропией Шеннона](#) и измеряет непредсказуемость реализации случайной величины. В оригинальном определении, правда, речь шла не о значениях случайной величины, а о символах (первичного) алфавита, так как Шенон придумал эту величину, занимаясь вопросами кодирования строк. Для данной задачи энтропия имеет вполне практический смысл — среднее количество битов, которое необходимо для кодирования одного символа сообщения при заданной частоте символов алфавита.

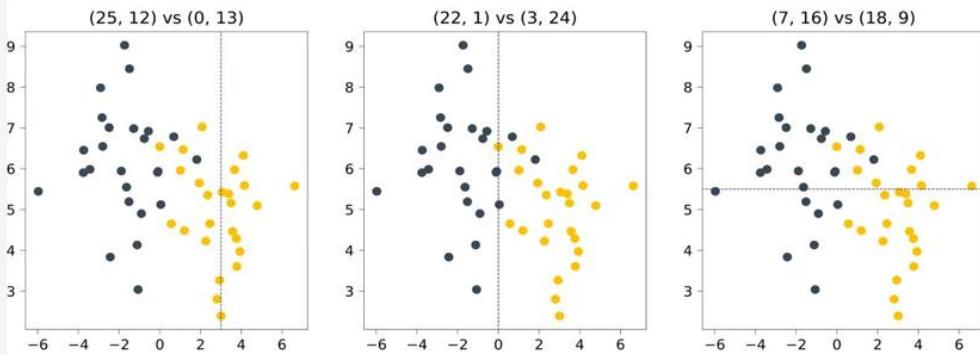
Так как $p_k \in [0, 1]$, энтропия неотрицательна. Если случайная величина принимает только одно значение, то она абсолютно предсказуема и её энтропия равна -1
 $\log(1) = 0$.

Наибольшего значения энтропия достигает для равномерно распределённой случайной величины — и это отражает тот факт, что среди всех величин с данной областью значений она наиболее «непредсказуема». Для равномерно распределённой на множестве $\{1, \dots, K\}$ случайной величины значение энтропии будет равно:

$$-\sum_{k=1}^K \frac{1}{K} \log \frac{1}{K} = \log K$$

На следующем графике приведены три дискретных распределения на множестве $\{0, 1, \dots, 20\}$ с их энтропиями. Как и указано выше, максимальную энтропию будет иметь равномерное распределение; у двух других проявляются пики разной степени остроты — и тем самым реализации этих величин обладают меньшей неопределенностью: мы можем с большей уверенностью говорить, что будет сгенерировано.

Разберём на примере игрушечной задачи классификации то, как энтропия может выступать в роли impurity. Рассмотрим три разбиения синтетического датасета и посмотрим, какие значения энтропии они дают. В подписях указано, каким становится соотношение классов в половинках.



В изначальном датасете по 25 точек каждого класса; энтропия состояния равна

$$S_0 = -\frac{25}{50} \log_2 \frac{25}{50} - \frac{25}{50} \log_2 \frac{25}{50} = 1$$

Для первого разбиения, по $[X_1 \leq 3]$ в левую часть попадают 25 точек класса 0 и 12 точек класса 1, а в правую — 0 точек класса 0 и 13 точек класса 1. Энтропия левой группы равна

Справа ноль

35. Проблема переобучения деревьев принятия решений. Регулирование глубины дерева (обрезка ветвей).36. Объясните понятие ошибок первого и второго рода, их связь с машинным обучением.

Предварительная обрезка, при которой дерево перестает расти раньше, прежде чем оно идеально классифицирует обучающий набор.

Пост-обрезка, которая позволяет дереву идеально классифицировать обучающий набор, а затем выполнять пост-обрезку дерева.

Практически, второй подход - переобученные деревья после обрезки - более успешен, потому что нелегко точно оценить, когда следует прекратить рост дерева. Важным этапом обрезки дерева является определение критерия, который будет использоваться для определения правильного окончательного размера дерева, используя один из следующих методов:

Первый метод - наиболее распространенный подход. В этом подходе доступные данные разделены на два набора примеров: обучающий набор, который используется для построения дерева решений, и набор проверки, который используется для оценки воздействия обрезки дерева. Второй способ - тоже распространенный подход. Здесь мы объясняем оценку ошибки и тест Chi2.

Post-pruning using Error estimation

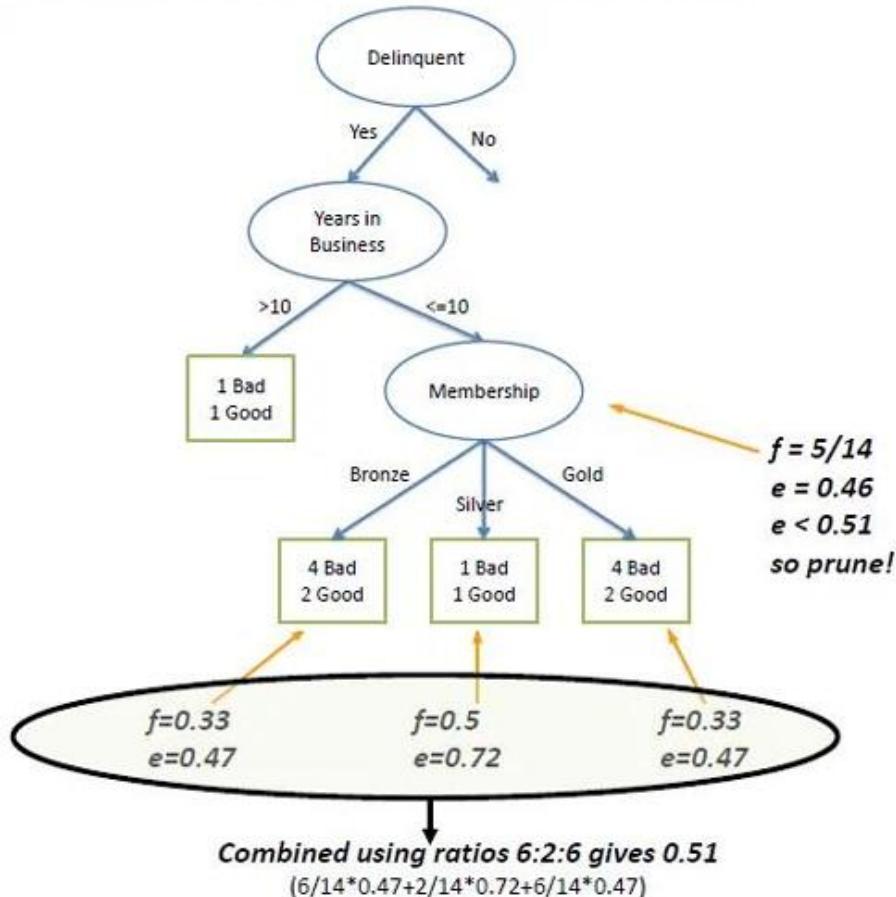
Error estimate for a sub-tree is weighted sum of error estimates for all its leaves. The error estimate (e) for a node is:

$$e = \left(f + \frac{z^2}{2N} + z \sqrt{\frac{f}{N} - \frac{f^2}{N} + \frac{z^2}{4N^2}} \right) \Bigg/ \left(1 + \frac{z^2}{N} \right)$$

Where:

- f is the error on the training data
- N is the number of instances covered by the leaf
- z from normal distribution

In the following example we set Z to 0.69 which is equal to a confidence level of 75%.



Частота ошибок в родительском узле составляет 0,46, и поскольку частота ошибок для его дочерних узлов (0,51) увеличивается с разбиением, мы не хотим оставлять дочерние узлы.

37. Accuracy, Recall, Precision, F1-мера

Эти метрики применяются для оценки качества классификации.

-Accuracy (точность общего попадания): доля всех правильных ответов (и положительных, и отрицательных) среди всех объектов. Хороша при сбалансированных классах.

-Precision (точность): доля реально положительных среди тех, что модель предсказала как положительные. Высокая precision означает мало ложных срабатываний.

-Recall (полнота): доля правильно найденных положительных объектов среди всех реально положительных. Высокая recall означает модель не пропускает.

-F1-мера: гармоническое среднее между precision и recall. Особенно полезна при несбалансированных классах и когда важен баланс между ошибками первого и второго рода.

Формулы:

- Accuracy = $(TP + TN) / (TP + TN + FP + FN)$
- Precision = $TP / (TP + FP)$
- Recall = $TP / (TP + FN)$
- $F1 = 2 \cdot (Precision \cdot Recall) / (Precision + Recall)$

Где:

TP — истинно положительные, FP — ложно положительные, TN — истинно отрицательные, FN — ложно отрицательные.

Вывод: выбор метрики зависит от задачи. Precision важна, если ошибки переполнения критичны (например, спам-фильтры), а recall — если важнее не пропустить (например, медицина). F1 подходит, когда нужен баланс.

38. ROC-кривая, PR-кривая, площадь под кривыми

ROC-кривая (Receiver Operating Characteristic):

График зависимости **полноты (TPR)** от **доли ложных срабатываний (FPR)** при разных порогах.

Показывает, насколько хорошо модель отделяет классы.

- $TPR = TP / (TP + FN)$
- $FPR = FP / (FP + TN)$

Площадь под ROC-кривой (AUC-ROC):

Характеризует качество ранжирования модели.

- AUC $\approx 1 \rightarrow$ идеальный классификатор
- AUC $\approx 0.5 \rightarrow$ случайные угадывания

PR-кривая (Precision-Recall):

График зависимости **точности от полноты**.

Полезнее при **несбалансированных классах**, когда положительный класс редок.

Площадь под PR-кривой (AUC-PR):

Показывает, насколько хорошо модель балансирует между precision и recall.

Чем выше AUC, тем лучше качество приоритетной выборки.

Вывод:

- ROC-кривая лучше при сбалансированных классах.
- PR-кривая — при редких положительных примерах.
- Площади под кривыми — это обобщённые оценки качества модели по всем порогам.

39. Метрики оценки качества регрессии

В задачах регрессии сравниваются предсказанные значения с реальными числовыми ответами с помощью следующих метрик:

- **MAE (средняя абсолютная ошибка):** среднее модулей разности между предсказаниями и реальными значениями.
→ Легко интерпретируется, устойчива к выбросам.
- **MSE (среднеквадратичная ошибка):** среднее квадратов отклонений.
→ Сильно штрафует большие ошибки, чувствительна к выбросам.
- **RMSE (корень из MSE):** корень из среднеквадратичной ошибки.
→ Имеет те же единицы измерения, что и целевая переменная.
- **R² (коэффициент детерминации):** показывает, какую долю дисперсии объясняет модель.
→ R² = 1 — идеальное предсказание, R² = 0 — модель не лучше среднего значения.

Вывод:

MAE показывает среднюю ошибку, MSE акцентирует большие отклонения, RMSE объединяет наглядность и строгость, а R² — относительное качество модели. Выбор метрики зависит от задачи и чувствительности к выбросам.

40. Задача кластеризации, типы кластеров, чувствительность к признакам (на русском)

Кластеризация — это задача без учителя, цель которой — разбить объекты на группы (кластеры) по сходству **без использования меток классов**. Сходство определяется на основе признаков.

Типы кластерных структур:

- **Шаровидные (выпуклые)**: кластеры круглые и отделяются по расстоянию (например, k-means).
- **Плотностные**: кластеры — плотные участки пространства (например, DBSCAN).
- **Иерархические**: кластеры вложены друг в друга (например, агломеративная кластеризация).
- **Произвольной формы**: методы, способные находить кластеры сложной формы.

Чувствительность к признакам:

Качество кластеризации **сильно зависит** от выбранных признаков и их масштабов. Лишние или нерелевантные признаки могут исказить структуру. Методы на основе расстояний (например, k-means) чувствительны к масштабам, поэтому **нормализация и отбор признаков** — критически важны.

Вывод: правильно подобранные признаки и масштабирование — ключ к качественной кластеризации.



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.

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:

$$F_0 = \frac{\sum_{i < j} [a_i = a_j] \rho(x_i, x_j)}{\sum_{i < j} [a_i = a_j]} \rightarrow \min.$$

- Среднее межкластерное расстояние:

$$F_1 = \frac{\sum_{i < j} [a_i \neq a_j] \rho(x_i, x_j)}{\sum_{i < j} [a_i \neq a_j]} \rightarrow \max.$$

- Отношение пары функционалов: $F_0/F_1 \rightarrow \min$.

Качество кластеризации в линейном векторном пространстве

Пусть объекты x_i задаются векторами $(f_1(x_i), \dots, f_n(x_i))$.

- Сумма средних внутрикластерных расстояний:

$$\Phi_0 = \sum_{a \in Y} \frac{1}{|X_a|} \sum_{i: a_i=a} \rho(x_i, \mu_a) \rightarrow \min,$$

$X_a = \{x_i \in X^\ell \mid a_i = a\}$ — кластер a ,
 μ_a — центр масс кластера a .

- Сумма межкластерных расстояний:

$$\Phi_1 = \sum_{a, b \in Y} \rho(\mu_a, \mu_b) \rightarrow \max.$$

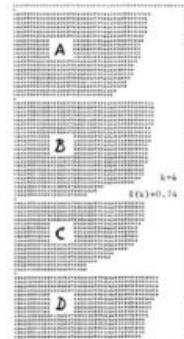
- Отношение пары функционалов: $\Phi_0 / \Phi_1 \rightarrow \min$.

Коэффициент силуэта (анализ ошибок кластеризации)

Распределение качества кластеризации по объектам/кластерам

- Ср.расстояние до объектов своего кластера:

$$r_i = \frac{1}{|X_{a_i}| - 1} \sum_{x \in X_{a_i} \setminus x_i} \rho(x, x_i)$$



- Мин. ср.расстояние до чужого кластера:

$$R_i = \min_{a \in Y \setminus a_i} \frac{1}{|X_a|} \sum_{x \in X_a} \rho(x, x_i)$$

- Коэффициент силуэта объекта: $s(i) = \frac{R_i - r_i}{\max(R_i, r_i)} \in [-1, +1]$

Peter J. Rousseeuw. Silhouettes: a graphical aid to the interpretation and validation of cluster analysis. 1987.

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:

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

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

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

E.Amigo, J.Gonzalo, J.Artiles, F.Verdejo. A comparison of extrinsic clustering evaluation metrics based on formal constraints. 2009.

43. Метод k-средних.

Минимизация суммы квадратов внутрикластерных расстояний:

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

Алгоритм Ллойда

вход: X^ℓ , $K = |Y|$; **выход:** центры кластеров μ_a , $a \in Y$;

$\mu_a :=$ начальное приближение центров, для всех $a \in Y$;

повторять

отнести каждый x_i к ближайшему центру:

$$a_i := \arg \min_{a \in Y} \|x_i - \mu_a\|, \quad i = 1, \dots, \ell;$$

вычислить новые положения центров:

$$\mu_a := \frac{\sum_{i=1}^{\ell} [a_i = a] x_i}{\sum_{i=1}^{\ell} [a_i = a]}, \quad a \in Y;$$

пока a_i не перестанут изменяться;

Метод K-средних (K-means) для частичного обучения

Модификация алгоритма Ллойда
при наличии размеченных объектов $\{x_1, \dots, x_k\}$

вход: $X^\ell, K = |Y|$;

выход: центры кластеров $\mu_a, a \in Y$;

$\mu_a :=$ начальное приближение центров, для всех $a \in Y$;

повторять

- отнести каждый $x_i \in U$ к ближайшему центру:
 $a_i := \arg \min_{a \in Y} \|x_i - \mu_a\|, i = k+1, \dots, \ell$;
- вычислить новые положения центров:
$$\mu_a := \frac{\sum_{i=1}^{\ell} [a_i = a] x_i}{\sum_{i=1}^{\ell} [a_i = a]}, a \in Y$$

пока a_i не перестанут изменяться;

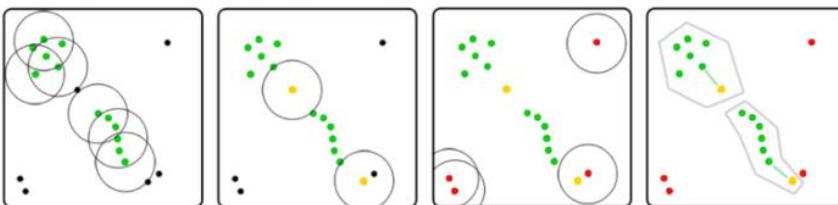
44. Алгоритм DBSCAN.

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



вход: выборка $X^\ell = \{x_1, \dots, x_\ell\}$; параметры ε и m ;

выход: разбиение выборки на кластеры и шумовые выбросы;

$U := X^\ell$ — непомеченные; $a := 0$;

пока в выборке есть непомеченные точки, $U \neq \emptyset$:

 взять случайную точку $x \in U$;

 если $|U_\varepsilon(x)| < m$ то

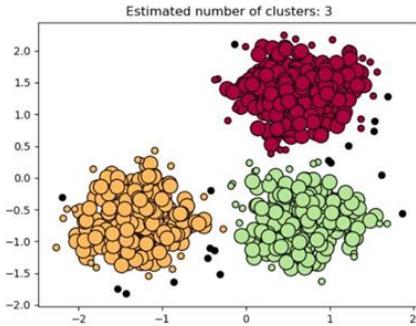
- пометить x как, возможно, шумовой;

 иначе

- создать новый кластер: $K := U_\varepsilon(x)$; $a := a + 1$;
- для всех $x' \in K$, не помеченных или шумовых
 - если $|U_\varepsilon(x')| \geq m$ то $K := K \cup U_\varepsilon(x')$;
 - иначе пометить x' как граничный кластера K ;
- $a_i := a$ для всех $x_i \in K$;

$U := U \setminus K$;

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



45. Иерархическая кластеризация.

Алгоритм иерархической кластеризации (Ланс, Уильямс, 1967):
итеративный пересчёт расстояний R_{UV} между кластерами U, V .

$C_1 := \{\{x_1\}, \dots, \{x_\ell\}\}$ — все кластеры 1-элементные;
 $R_{\{x_i\}\{x_j\}} := \rho(x_i, x_j)$ — расстояния между ними;
для всех $t = 2, \dots, \ell$ (t — номер итерации):
 | найти в C_{t-1} пару кластеров (U, V) с минимальным R_{UV} ;
 | слить их в один кластер:
 | $W := U \cup V$;
 | $C_t := C_{t-1} \cup \{W\} \setminus \{U, V\}$;
 | **для всех** $S \in C_t$
 | | вычислить R_{WS} по формуле Ланса-Уильямса:
 | | $R_{WS} := \alpha_U R_{US} + \alpha_V R_{VS} + \beta R_{UV} + \gamma |R_{US} - R_{VS}|$;

Алгоритм Ланса-Уильямса для частичного обучения

Алгоритм иерархической кластеризации (Ланс, Уильямс, 1967):
итеративный пересчёт расстояний R_{UV} между кластерами U, V .

$C_1 := \{\{x_1\}, \dots, \{x_\ell\}\}$ — все кластеры 1-элементные;
 $R_{\{x_i\}\{x_j\}} := \rho(x_i, x_j)$ — расстояния между ними;
для всех $t = 2, \dots, \ell$ (t — номер итерации):
 | найти в C_{t-1} пару кластеров (U, V) с минимальным R_{UV} ,
 | | **при условии, что в $U \cup V$ нет объектов с разными метками**;
 | слить их в один кластер:
 | $W := U \cup V$;
 | $C_t := C_{t-1} \cup \{W\} \setminus \{U, V\}$;
 | **для всех** $S \in C_t$
 | | вычислить R_{WS} по формуле Ланса-Уильямса:
 | | $R_{WS} := \alpha_U R_{US} + \alpha_V R_{VS} + \beta R_{UV} + \gamma |R_{US} - R_{VS}|$;

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

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

R^U не монотонно; R^B, R^A, R^G, R^Y — монотонны.

R^B — сжимающее; R^A, R^Y — растягивающие;

46. Карты Кохонена.

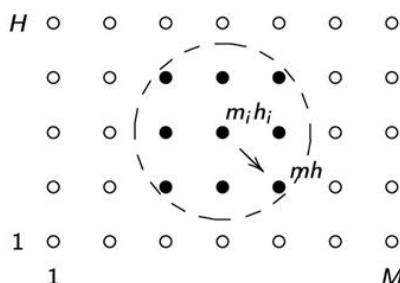
$Y = \{1, \dots, M\} \times \{1, \dots, H\}$ — прямоугольная сетка кластеров

Каждому узлу (m, h) приписан нейрон Кохонена $\theta_{mh} \in \mathbb{R}^n$

Наряду с метрикой $\rho(x_i, x)$ на X вводится метрика на сетке Y :

$$r((m_i, h_i), (m, h)) = \sqrt{(m - m_i)^2 + (h - h_i)^2}$$

Окрестность (m_i, h_i) :



Вход: X^ℓ — обучающая выборка; η — темп обучения;

Выход: $\theta_{mh} \in \mathbb{R}^n$ — векторы весов, $m = 1..M, h = 1..H$;

$\theta_{mh} := \text{random}\left(-\frac{1}{2MN}, \frac{1}{2MN}\right)$ — инициализация весов;

повторять

выбрать объект x_i из X^ℓ случайным образом;

WTA: вычислить координаты кластера:

$$(m_i, h_i) := g(x_i) \equiv \arg \min_{(m, h) \in Y} \rho(x_i, \theta_{mh})$$

для всех $(m, h) \in \text{Окрестность}(m_i, h_i)$

WTM: сделать шаг градиентного спуска:

$$\theta_{mh} := \theta_{mh} + \eta(x_i - \theta_{mh})K(r((m_i, h_i), (m, h)))$$

пока кластеризация не стабилизируется;

Интерпретация карт Кохонена

Два типа графиков — цветных карт $M \times H$:

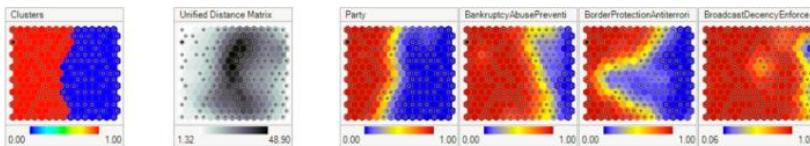
- Цвет узла (m, h) — локальная плотность в точке (m, h) — среднее расстояние до k ближайших точек выборки
- По одной карте на каждый признак:
цвет узла (m, h) — значение j -й компоненты вектора $w_{m,h}$

Пример: задача UCI house-votes (US Congress voting patterns)

Объекты — конгрессмены

Признаки — результаты голосования по различным вопросам

Есть целевой признак «партия» $\in \{\text{демократ, республиканец}\}$



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.

Определение ансамбля

$X^\ell = (x_i, y_i)_{i=1}^\ell \subset X \times Y$ — обучающая выборка, $y_i = y^*(x_i)$

$g_t: X \rightarrow Y, t = 1, \dots, T$ — обучаемые базовые алгоритмы

Идея ансамблирования: как из множества по отдельности плохих алгоритмов g_t построить один хороший?

Декомпозиция базовых алгоритмов $g_t(x) = C(b_t(x))$

$g_t: X \xrightarrow{b_t} R \xrightarrow{C} Y$, где R — пространство оценок

b_t — алгоритмические операторы, C — решающее правило

Ансамбль (композиция) базовых алгоритмов g_1, \dots, g_T ,

$F: F^T \rightarrow R$ — корректирующая (агрегирующая) операция

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

Агрегирующие (корректирующие) функции

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.

Способы повышения разнообразия с помощью рандомизации:

- bagging (bootstrap aggregating) — подвыборки обучающей выборки «с возвращением», в каждую выборку попадает $1 - (1 - \frac{1}{\ell})^\ell \rightarrow 1 - \frac{1}{e} \approx 63.2\%$ объектов, при $\ell \rightarrow \infty$
- pasting — случайные обучающие подвыборки
- random subspaces — случайные подмножества признаков
- random patches — случ. подмн-ва и объектов, и признаков
- cross-validated committees — выборка разбивается на k блоков (k -fold) и делается k обучений без одного блока

Вход: обучающая выборка X^ℓ ; параметры: T ,

ℓ' — объём обучающих подвыборок,

n' — размерность признаковых подпространств,

ε_1 — порог качества базовых алгоритмов на обучении,

ε_2 — порог качества базовых алгоритмов на контроле;

Выход: базовые алгоритмы b_t , $t = 1, \dots, T$;

для всех $t = 1, \dots, T$:

$U_t :=$ случайная подвыборка объёма ℓ' из X^ℓ ;

$G_t :=$ случайное подмножество мощности n' из F^n ;

$b_t := \mu(G_t, U_t)$;

если $Q(b_t, U_t) > \varepsilon_1$ то не включать b_t в ансамбль;

если $Q(b_t, X^\ell \setminus U_t) > \varepsilon_2$ то не включать b_t в ансамбль;

bagging
+
random subspaces

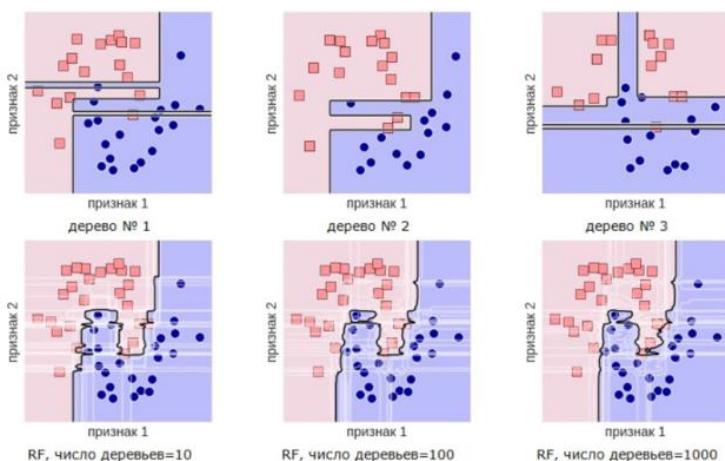
Обучение случайного леса:

- 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

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 (Difference between bagging and boosting)

Идея метода

Бэггинг (bagging)

- Обучаем несколько классификаторов независимо
- Итоговое решение — голосование или усреднение ответов

Бустинг (boosting)

- Обучаем классификаторы последовательно
- Каждый следующий исправляет ошибки предыдущего
- Итог — взвешенная комбинация слабых алгоритмов

Бэггинг на подпространствах

- Для классификации объектов $(x^{(1)}, \dots, x^{(m)})$:
 - каждое $x^{(i)}$ классифицируется независимо
 - итог — голосование по всем $x^{(i)}$
- Бэггинг можно рассматривать как классификацию «объекта-подпространства»

Бустинг

- Итеративно строится композиция алгоритмов:

$$a_k = \text{sign} \left(\sum_{t=1}^k \alpha_t b_t(x) \right)$$

где α_t — вес t -го классификатора, b_t — t -й базовый классификатор

51. Алгоритм AdaBoost.

Вход: обучающая выборка X^ℓ ; параметр T ;

Выход: базовые алгоритмы и их веса $\alpha_t b_t$, $t = 1, \dots, T$;

инициализировать веса объектов: $w_i := 1/\ell$, $i = 1, \dots, \ell$;

для всех $t = 1, \dots, T$:

обучить базовый алгоритм:

$$b_t := \arg \min_b N(b; W^\ell);$$

$$\alpha_t := \frac{1}{2} \ln \frac{1 - N(b_t; W^\ell)}{N(b_t; W^\ell)};$$

обновить веса объектов:

$$w_i := w_i \exp(-\alpha_t y_i b_t(x_i)), \quad i = 1, \dots, \ell;$$

нормировать веса объектов:

$$w_0 := \sum_{j=1}^{\ell} w_j;$$

$$w_i := w_i / w_0, \quad i = 1, \dots, \ell;$$

Вход: обучающая выборка X^ℓ ; **параметр T** ;

Выход: базовые алгоритмы и их веса $\alpha_t b_t$, $t = 1, \dots, T$;

инициализация: $a_{0,i} := 0$, $i = 1, \dots, \ell$;

для всех $t = 1, \dots, T$

базовый алгоритм, приближающий антиградиент:

$$b_t := \arg \min_{b \in \mathcal{B}} \sum_{i=1}^{\ell} (b(x_i) + \mathcal{L}'(a_{t-1,i}, y_i))^2;$$

задача одномерной минимизации:

$$\alpha_t := \arg \min_{\alpha > 0} \sum_{i=1}^{\ell} \mathcal{L}(a_{t-1,i} + \alpha b_t(x_i), y_i);$$

обновление вектора значений на объектах выборки:

$$a_{t,i} := a_{t-1,i} + \alpha_t b_t(x_i); \quad i = 1, \dots, \ell;$$

Стохастический градиентный бустинг

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

Преимущества:

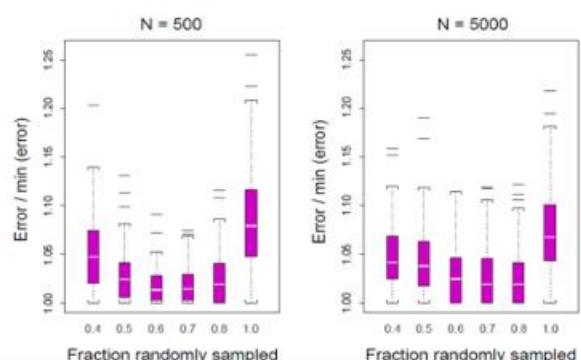
- улучшается сходимость, уменьшается время обучения
- улучшается обобщающая способность ансамбля
- можно использовать несмешённые оценки out-of-bag

Эксперименты:

относительная ошибка при различном объёме выборки N

Вывод:

оптимально сэмплировать около 60–80% выборки



53. Алгоритмы CatBoost и XGBoost (на русском)

XGBoost (Extreme Gradient Boosting):

XGBoost — это мощная и масштабируемая реализация градиентного бустинга. Он строит модель **поэтапно**, где каждое следующее дерево подгоняется к **остаткам (ошибкам)** текущей модели. Использует **второй порядок разложения функции потерь** (градиенты и гессианы), что делает обновления более точными. Ключевые особенности:

- **Регуляризация (L1 и L2)** помогает бороться с переобучением
 - **Параллельное обучение** по блокам признаков
 - **Устойчивость к пропущенным значениям**
 - **Обрезка деревьев** по максимальной глубине или убыточности
 - Поддержка пользовательских функций потерь и ранней остановки
- XGBoost — выбор №1 на Kaggle, особенно на числовых табличных данных.

CatBoost (Categorical Boosting):

CatBoost — это библиотека градиентного бустинга от Яндекса, ориентированная на работу с **категориальными признаками**. Использует метод **упорядоченного бустинга**, который избегает утечки целевой переменной путём учета порядка обучения. В отличие от XGBoost, CatBoost **не требует ручного кодирования категорий** — он делает это автоматически, оптимальным образом. Главные преимущества:

- **Обработка категориальных признаков без препроцессинга**
- **Симметричные деревья** (одинаковая структура), что ускоряет предсказания
- **Устойчивость к переобучению** за счёт oblivious-структуры деревьев и встроенной регуляризации
- **Поддержка GPU** и встроенная кросс-валидация

Когда использовать:

- **CatBoost** лучше, если в данных много категориальных переменных и важна простота
- **XGBoost** лучше при числовых данных, когда важна высокая настраиваемость и производительность

Вывод: оба алгоритма — лучшие инструменты для задач машинного обучения на табличных данных. Выбор зависит от структуры данных и целей проекта.

54. Работа с большими данными. Экосистема Apache Hadoop

Big Data (большие данные) — это данные, которые слишком объёмны, разнообразны или быстро поступают, чтобы их можно было обработать обычными средствами. Для работы с такими данными нужны распределённые системы.

Apache Hadoop — это open-source платформа для надёжной и масштабируемой обработки больших данных на кластере из множества машин.

Основные компоненты Hadoop:

- HDFS (распределённая файловая система Hadoop): делит файлы на блоки, распределяет их по узлам кластера и дублирует для отказоустойчивости.
- MapReduce: модель распределённой обработки — этап Map разбивает задачи, а Reduce агрегирует результаты.
- YARN: управляет ресурсами и распределяет задачи по узлам кластера.

Инструменты экосистемы Hadoop:

- Hive: язык запросов, похожий на SQL, для анализа данных в HDFS.
- Pig: язык сценариев для пакетной обработки больших объёмов данных.
- HBase: NoSQL база данных поверх HDFS, обеспечивает быстрый доступ к данным.
- Sqoop: импорт и экспорт данных между Hadoop и реляционными БД.
- Flume: сбор и передача больших потоков логов.
- Oozie: система управления задачами и расписаниями внутри Hadoop.

Преимущества Hadoop:

- Линейное масштабирование — можно просто добавлять узлы
- Устойчивость к сбоям — благодаря дублированию данных
- Идеален для пакетной обработки (но не для реального времени)

Вывод: Apache Hadoop — основа для работы с большими данными, объединяющая хранилище, обработку и управление в распределённой среде.

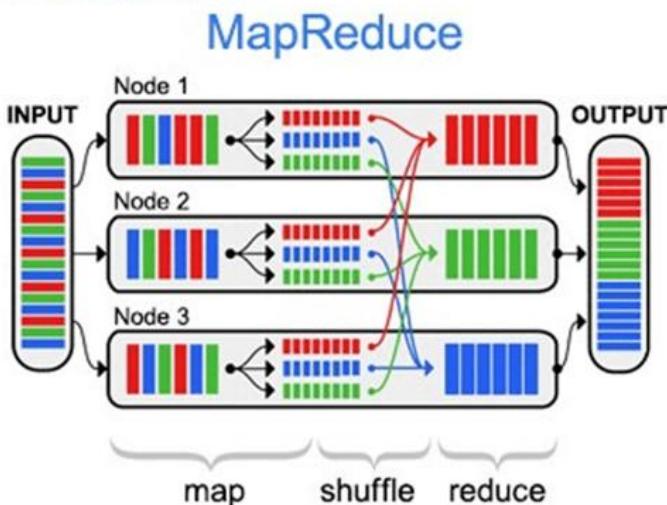
55. Файловая система HDFS.

1. Управляющий узел, узел имен или сервер имен (**NameNode**) – отдельный, единственный в кластере, сервер с программным кодом для управления пространством имен файловой системы, хранящий дерево файлов, а также метаданные файлов и каталогов. **NameNode** – обязательный компонент кластера HDFS, который отвечает за открытие и закрытие файлов, создание и удаление каталогов, управление доступом со стороны внешних клиентов и соответствие между файлами и блоками, дублированными (репликованными) на узлах данных. Сервер имён раскрывает для всех желающих расположение блоков данных на машинах кластера.
2. Secondary NameNode – вторичный узел имен, отдельный сервер, единственный в кластере, который копирует образ HDFS и лог транзакций операций с файловыми блоками во временную папку, применяет изменения, накопленные в логе транзакций к образу HDFS, а также записывает его на узел NameNode и очищает лог транзакций. Secondary NameNode необходим для быстрого ручного восстановления NameNode в случае его выхода из строя.
3. Узел или сервер данных (**DataNode**, **Node**) – один из множества серверов кластера с программным кодом, отвечающим за файловые операции и работу с блоками данных. **DataNode** – обязательный компонент кластера HDFS, который отвечает за запись и чтение данных, выполнение команд от узла NameNode по созданию, удалению и репликации блоков, а также периодическую отправку сообщения о состоянии (heartbeats) и обработку запросов на чтение и запись, поступающих от клиентов файловой системы HDFS. Стоит отметить, что данные проходят с остальных узлов кластера к клиенту мимо узла NameNode.
4. Клиент (**client**) – пользователь или приложение, взаимодействующий через специальный интерфейс (API – Application Programming Interface) с распределенной файловой системой. При наличии достаточных прав, клиенту разрешены следующие операции с файлами и каталогами: создание, удаление, чтение, запись, переименование и перемещение. Создавая файл, клиент может явно указать размер блока файла (по умолчанию 64 Мб) и количество создаваемых реплик (по умолчанию значение равно 3-ем).

Благодаря репликации блоков по узлам данных, распределенная файловая система Hadoop обеспечивает высокую надежность хранения данных и скорость вычислений. Кроме того, **HDFS обладает рядом отличительных свойств [4]:**

- ✓ **большой размер блока** по сравнению с другими файловыми системами (>64МВ), поскольку HDFS предназначена для хранения большого количества огромных (>10ГБ) файлов;
- ✓ **ориентация на недорогие и, поэтому не самые надежные сервера** – отказоустойчивость всего кластера обеспечивается за счет репликации данных;
- ✓ **зеркалирование и репликация** осуществляются на уровне кластера, а не на уровне узлов данных;
- ✓ **репликация происходит в асинхронном режиме** – информация распределяется по нескольким серверам прямо во время загрузки, поэтому выход из строя отдельных узлов данных не повлечет за собой полную пропажу данных;
- ✓ **HDFS оптимизирована для потоковых считываний файлов**, поэтому применять ее для нерегулярных и произвольных считываний нецелесообразно;
- ✓ **клиенты могут считывать и писать** файлы HDFS напрямую через программный интерфейс Java;
- ✓ **файлы пишутся однократно**, что исключает внесение в них любых произвольных изменений;
- ✓ **принцип WORM (Write-once and read-many, один раз записать** – много раз прочитать) полностью освобождает систему от блокировок типа «запись-чтение». Запись в файл в одно время доступен только одному процессу, что исключает конфликты множественной записи.
- ✓ **HDFS оптимизирована под потоковую передачу данных;**
- ✓ **сжатие данных и рациональное использование дискового пространства** позволило снизить нагрузку на каналы передачи данных, которые чаще всего являются узким местом в распределенных средах;
- ✓ **самодиагностика** – каждый узел данных через определенные интервалы времени отправляет диагностические сообщения узлу имен, который записывает логи операций над файлами в специальный журнал;
- ✓ **все метаданные сервера имен хранятся в оперативной памяти.**

56. Алгоритм Map-Reduce.



MapReduce – это модель распределённых вычислений от компании Google, используемая в технологиях [Big Data](#) для параллельных вычислений над очень большими (до нескольких петабайт) наборами данных в компьютерных кластерах, и фреймворк для вычисления распределенных задач на узлах (node) кластера [1].

Технология практически универсальна: она может использоваться для индексации веб-контента, подсчета слов в большом файле, счётчиков частоты обращений к заданному адресу, вычисления объёма всех веб-страниц с каждого URL-адреса конкретного хост-узла, создания списка всех адресов с необходимыми данными и прочих задач обработки огромных массивов распределенной информации. Также к областям применения MapReduce относится распределённый поиск и сортировка данных, обращение графа веб-ссылок, обработка статистики логов сети, построение инвертированных индексов, кластеризация документов, машинное обучение и статистический машинный перевод. Также MapReduce адаптирована под многопроцессорные системы, добровольные вычислительные, динамические облачные и мобильные среды [2].

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

Apache Spark

универсальная кластерная вычислительная платформа.

Существенно более высокая скорость для ряда задач по сравнению со стандартным MapReduce.

API на Java, Scala, Python

Resilient distributed datasets (RDD) - коллекции элементов, распределенных между множеством вычислительных узлов, которые могут обрабатываться параллельно.

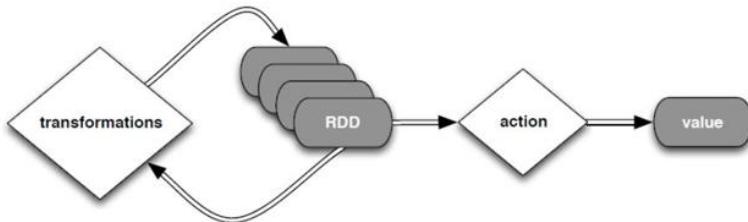
RDD

Два типа наборов данных:

parallelized collections –
создаются на базе существующих коллекций в оперативной памяти,
реализуется параллельная обработка.

Hadoop datasets – операции выполняются над каждой записью в файле HDFS или в другом хранилище, поддерживаемом Hadoop.

Преобразования и операции



Функции преобразования задают преобразования, которые будут применяться при выполнении операций

типы преобразований

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)	$(K,V1), (K,V2) \rightarrow (K, \text{seq } (V1 \text{ seq } (V2)))$

U — множество клиентов (субъектов/пользователей — users)

I — множество объектов (товаров/предметов — items)

Типы исходных данных:

- $D = (u_t, i_t, y_t)_{t=1}^T \in U \times I \times Y$ — транзакционные данные,
 Y — пространство описаний транзакций
- $R = (r_{ui})_{U \times I}$ — матрица отношений (или кросс-табуляции)
 $r_{ui} = \text{aggr}\{(u_t, i_t, y_t) \in D \mid u_t = u, i_t = i\}$
- $r_{ui} \in \{0, 1\}$ — бинарные данные
- $r_{ui} \in \{1, 2, \dots, M\}$ — рейтинги (порядковые или целые)

Задачи в рекомендательных системах:

- *заполнение пропусков* (missing values) в ячейках r_{ui}
- *ранжирование* списка top- n рекомендаций для u или для i

U — пользователи Интернет

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

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

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

$n_{iw} = \text{частота слова } w \text{ в документе } i$

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

Это два основных подхода в рекомендательных системах, помогающих предлагать пользователю релевантные товары, фильмы и т.п.

Коллаборативная фильтрация (CF):

Основывается на поведении пользователей — рекомендует, глядя на сходства между пользователями или товарами.

-User-based: «Пользователи, похожие на вас, оценили...»

-Item-based: «Те, кто смотрел этот фильм, также смотрели...»

Плюсы:

-Не требует знаний о содержимом товара

-Находит нестандартные связи

Минусы:

-Проблема холодного старта (новые пользователи/товары)

-Разреженность матрицы оценок

Контентная фильтрация (CBF):

Учитывает свойства объектов и предпочтения пользователя. Рекомендует похожие на ранее понравившиеся объекты (жанр, описание и т.д.).

Пример: «Вы смотрели боевики — вот ещё боевики».

Плюсы:

-Хорошо работает для новых пользователей

-Учитывает личные интересы

Минусы:

-Требует описаний объектов

-Может быть слишком узкой (мало разнообразия)

Вывод:

Обе модели имеют сильные и слабые стороны, поэтому на практике часто используют гибридные системы, совмещающие колаборативный и контентный подход.

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

Коллаборативная фильтрация реализуется двумя основными способами:

1. Memory-Based (на основе памяти):

Использует исходную матрицу взаимодействий «пользователь–объект» без обучения модели.

-User-based: рекомендует, что понравилось похожим пользователям

-Item-based: рекомендует похожие на ранее понравившиеся объекты

-Метрики схожести: косинусное сходство, корреляция Пирсона

-Простая реализация и интерпретация

-Плохо масштабируется, чувствительна к разреженности данных

2. Model-Based (на основе моделей):

Строит математическую модель на основе данных взаимодействий.

Примеры:

-Матричная факторизация (SVD и др.): выявляет скрытые характеристики пользователей и объектов

-Нейросети: выявляют сложные зависимости

-Кластеризация, регрессия, вероятностные модели

Плюсы:

-Подходит для больших и разреженных данных

-Часто точнее, чем memory-based

-Меньше интерпретируемость, но лучше обобщение

Вывод:

-Memory-based — быстро и просто, но плохо масштабируется

-Model-based — более сложно, но эффективнее на реальных и больших данных

61. Корреляционные модели в коллаборативной фильтрации.

Непараметрическая регрессия, функции сходства.

Непараметрическая регрессия для восстановления пропусков

Оценка рейтинга по схожим клиентам (User-Based CF):

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

$U(u)$ — коллаборация, множество клиентов, схожих с u

\bar{r}_u — средний рейтинг клиента u

$S(u, v)$ — функция сходства пары клиентов (u, v)

Оценка рейтинга по схожим объектам (Item-Based CF):

$$\hat{r}_{ui} = \bar{r}_i + \frac{\sum_{j \in I(i)} S(i, j)(r_{uj} - \bar{r}_j)}{\sum_{j \in I(i)} S(i, j)}$$

$I(i)$ — множество объектов, схожих с i

\bar{r}_i — средний рейтинг объекта i

$S(i, j)$ — функции сходства пары объектов (i, j)

ФУНКЦИИ СХОДСТВА ДЛЯ РЕЙТИНГОВЫХ ДАННЫХ

$I(u)$ — множество объектов, которые клиент u отрейтинговал
 $I(u, v)$ — множество объектов, которые отрейтинговали u и v

Корреляция Пирсона:

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

Косинусная мера сходства:

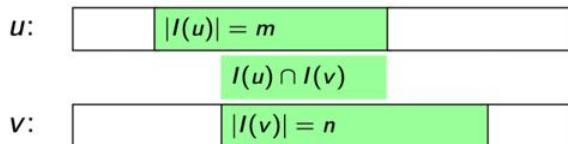
$$S(u, v) = \frac{\sum_{i \in I(u, v)} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I(u)} r_{ui}^2} \sqrt{\sum_{i \in I(v)} r_{vi}^2}},$$

где неявно предполагается, что $r_{ui} = 0$, если $i \notin I(u)$

Функция сходства $S(i, j)$ пар объектов определяется аналогично

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:

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

62. Понятие латентной модели.

Латентная модель: по данным D оцениваются векторы:

$(p_{tu})_{t \in G}$ — профили клиентов $u \in U$, $|G| \ll |I|$

$(q_{ti})_{t \in H}$ — профили объектов $i \in I$, $|H| \ll |U|$

Типы латентных моделей (основные идеи):

- Ко-кластеризация:
 - жёtsкая: $\begin{cases} p_{tu} = [\text{клиент } u \text{ принадлежит кластеру } t \in G]; \\ q_{ti} = [\text{объект } i \text{ принадлежит кластеру } t \in H]; \end{cases}$
 - мягкая: p_{tu}, q_{ti} — степени принадлежности кластерам.
- Матричные разложения: $G \equiv H$ — множество тем; по p_{tu}, q_{ti} должны восстанавливаться r_{ui} .
- Вероятностные модели: $G \equiv H$ — множество тем;
 $p_{tu} = p(t|u)$, $q_{ti} = q(t|i)$

63. Матричные разложения. Сингулярное разложение.

Сингулярное разложение (singular value decomposition, SVD)

Постановка задачи: $\|R - PQ^T\|^2 \rightarrow \min_{P, Q}$

Сингулярное разложение: $\hat{R} = \underbrace{V \sqrt{D}}_P \underbrace{\sqrt{D} U^T}_{Q^T}, U^T U = I, V^T V = I$

Достоинства:

- можно применять готовые библиотеки линейной алгебры
- хорошее ранжирование предложений на некоторых данных

Недостатки:

- если r_{ui} не известно, то полагаем $r_{ui} = 0$
(неявно считаем, что если клиент u никогда не выбирал объект i , то он ему, скорее всего, не интересен)
- ортогональность (собственных) векторов p_t, q_t
- неинтерпретируемость компонент векторов p_u, q_i

Cremonesi P., Koren Y., Turrin R. Performance of recommender algorithms on top-n recommendation tasks. RecSys 2010.

64. Измерение качества рекомендаций.

Измерение качества рекомендаций

RMSE — точность предсказания рейтингов (как в NetflixPrize):

$$\text{RMSE}^2 = \sum_{(u,i) \in D} (r_{ui} - \hat{r}_{ui})^2$$

Точность предсказаний не гарантирует хороших рекомендаций

$R_u(k) \subset I$ — первые k рекомендаций для клиента u

$L_u \subset I$ — истинные предпочтения клиента u

Более адекватные метрики качества рекомендаций:

- precision@ $k = \frac{|R_u(k) \cap L_u|}{|R_u(k)|}$ — точность
- recall@ $k = \frac{|R_u(k) \cap L_u|}{|L_u|}$ — полнота
- меры качества ранжирования: MAP, NDCG и др.

65. Понятие искусственного нейрона.

65. Искусственный нейрон и оценка качества рекомендаций (на русском)

Искусственный нейрон:

Искусственный нейрон — это базовый элемент нейросети, вдохновлённый работой биологических нейронов. Он принимает несколько входных значений (признаков), умножает их на веса, складывает, добавляет смещение (*bias*), и передаёт результат через функцию активации (например, sigmoid или ReLU). Он "активируется", если входной сигнал достаточно силён. Используется в построении нейронных сетей.

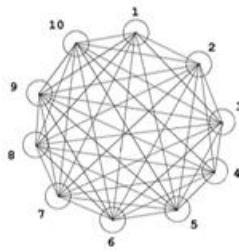
Оценка качества рекомендаций:

1. **Разнообразие (diversity):** Насколько сильно рекомендации отличаются между собой (по категориям, стилям и т.д.), и насколько они разнообразны от сессии к сессии у одного пользователя.
2. **Новизна (novelty):** Сколько среди рекомендаций объектов, которых пользователь ещё не видел. Повышает интерес и способствует открытию нового.
3. **Покрытие (coverage):** Какая доля всех доступных объектов когда-либо рекомендовалась кому-либо. Высокое покрытие означает лучшее использование базы контента.
4. **Догадливость (serendipity):** Умение рекомендательной системы угадывать нетривиальные и приятные сюрпризы — то, что пользователь не ожидал, но оценил.
5. **Предвзятость (propensity):** Смещение рекомендаций из-за истории поведения пользователя. Система может «зациклиться», предлагая одно и то же.

Важно: Обычно оптимизируют линейную комбинацию этих метрик или фокусируются на одной из них, накладывая ограничения на остальные (например, максимизировать новизну при минимальном уровне точности).

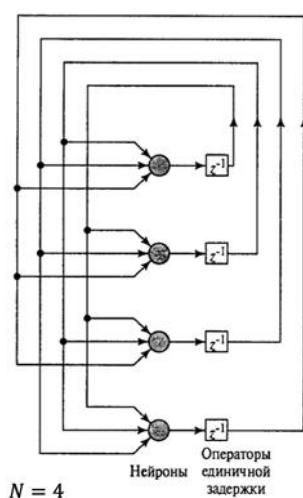
66. Сеть Хопфилда.

- 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



- Состояние сети (модели) из N нейронов:
 $y = (y_1, y_2, \dots, y_N)^T$

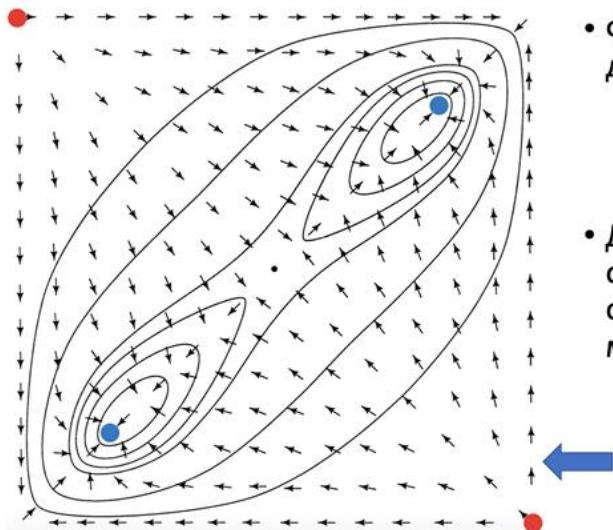
- Дискретная сеть Хопфилда:

$$y_j = \text{sign } u_j = \begin{cases} +1, u_j > 0 \\ -1, u_j < 0 \end{cases}$$

$$u_j = \sum_{i=1}^N w_{ji} y_i + b_j$$

где $w_{ji} = \theta_{ji}$ – параметры модели Хопфилда, $w_{ii} = 0$.

- Непрерывная сеть Хопфилда: система ДУ 1-го порядка, функция активации – сигмоида



- 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 .

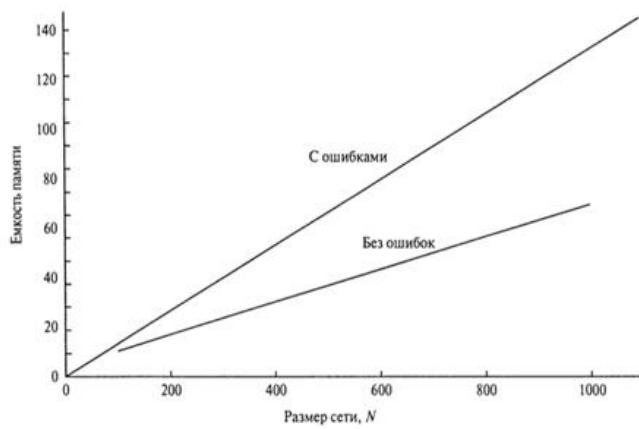
Контурная карта энергии непрерывной модели Хопфилда с двумя нейронами и двумя устойчивыми состояниями. Ординаты и абсциссы являются выходами двух нейронов. Устойчивые состояния соответствуют минимумам энергии и показаны синим, неустойчивые экстремумы – красным.

- 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 x_k из обучающей выборки

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:
$$\ell = \frac{N}{2 \ln N}$$

66. Понятие стохастического нейрона. Машина Больцмана.

Стохастический нейрон:

В отличие от обычного (детерминированного) нейрона, стохастический нейрон выдаёт случайный результат (0 или 1) в зависимости от входного сигнала. Чем сильнее вход, тем выше вероятность, что нейрон "сработает".

Обычно вероятность активации задаётся **сигмоидой**. Такая случайность позволяет исследовать разные состояния сети, что полезно при **обучении без учителя**.

Машина Больцмана:

Это разновидность **стохастической рекуррентной нейросети**, предназначенная для обучения без учителя. Сеть состоит из бинарных стохастических нейронов, соединённых симметрично.

- Обучение основано на **энергетической функции**: сеть стремится к состоянию с минимальной энергией.
- Каждый нейрон случайно обновляет своё состояние в зависимости от соседей.
- В результате сеть приходит в устойчивое состояние, отражающее закономерности в данных.
- Цель — уменьшить энергию наблюдаемых (правильных) состояний и повысить энергию случайных.

Ограниченнная машина Больцмана (RBM):

Упрощённая модель с двумя слоями: **видимый и скрытый, без связей внутри слоя.**

Часто используется в **глубоком обучении** для извлечения признаков и снижения размерности.

Вывод: стохастические нейроны и машины Больцмана моделируют вероятностную природу обучения, особенно полезны в задачах кластеризации, генерации данных и глубоком обучении.

68. Алгоритм имитации отжига (на русском)

Имитация отжига — это **вероятностный алгоритм оптимизации**, основанный на физическом процессе отжига металлов, при котором вещество медленно охлаждается, достигая состояния с минимальной энергией (оптимального).

Цель: Найти **глобальный минимум** функции, которая имеет много локальных минимумов.

Принцип работы:

1. Начинаем с случайного решения.
2. Вносим небольшое случайное изменение (соседнее решение).
3. Если новое решение лучше — принимаем его.
4. Если хуже — принимаем с некоторой вероятностью, зависящей от ухудшения и текущей температуры.
5. Температура постепенно понижается.
6. Процесс повторяется, пока система не "замёрзнет".

Вероятность приёма плохих решений:

При **высокой температуре** система активно исследует пространство решений, при **низкой** — действует более жадно, улучшая локальные решения.

Применяется для:

Комбинаторной оптимизации (например, задача коммивояжёра, расписания), настройки гиперпараметров, выхода из локальных минимумов.

Вывод: алгоритм имитации отжига сочетает случайность и постепенное "охлаждение", что позволяет находить качественные решения в сложных задачах.

69. Ограниченная машина Больцмана (RBM) и алгоритм контрастного расхождения (на русском)

Ограниченная машина Больцмана (RBM):

RBM — это стохастическая нейросеть с двумя слоями:

- **Видимый слой:** содержит входные данные
- **Скрытый слой:** обучается выявлять скрытые признаки
- **Нет связей внутри одного слоя,** только между слоями

Каждый нейрон — бинарный и случайный. Модель обучается путём минимизации **энергетической функции**, чтобы научиться аппроксимировать распределение входных данных. RBM используется для **извлечения признаков, предобучения и в генеративных моделях.**

Алгоритм контрастного расхождения (Contrastive Divergence, CD):

Это **быстрый приближённый метод** обучения RBM, потому что точный градиент правдоподобия вычислять слишком затратно. CD использует мало шагов выборки, чтобы оценить направление градиента.

Как работает:

1. Подаём реальные данные на видимый слой.
2. Вычисляем активации скрытого слоя.
3. Реконструируем вход на основе скрытого слоя.
4. Сравниваем оригинал и реконструкцию — **контраст** между ними даёт поправку к весам.
5. Повторяем с мини-батчами.

Этот контраст между реальными и восстановленными данными даёт направление для обновления весов.

Вывод: RBM — простая, но мощная модель для обучения без учителя, а CD делает её обучение быстрым и практически применимым.

70. Глубокие сети на основе машины Больцмана (на русском)

Глубокие доверительные сети (Deep Belief Networks, DBN) — это глубокие нейросети, построенные путём **послойного соединения нескольких ограниченных машин Больцмана (RBM)**. Каждая RBM изучает уровни признаков выше предыдущего слоя.

Структура:

- Каждый слой — это RBM.
- Видимый слой — это входные данные.
- Первая RBM изучает базовые признаки.
- Вторая RBM обучается на выходе первой и изучает более абстрактные признаки, и так далее.
- Последний слой может подключаться к классификатору или декодеру.

Обучение (жадное, послойное):

1. Обучаем первую RBM на исходных данных (метод контрастного расхождения).
2. Замораживаем веса и используем скрытый слой как вход для следующей RBM.
3. Повторяем для всех слоёв.
4. При необходимости — дообучаем всю сеть методом обратного распространения (если задача с учителем).

Применение:

- Обучение признакам без разметки
- Снижение размерности
- Предобучение глубоких сетей
- Генерация новых данных

Вывод: DBN — это способ построения глубоких нейросетей на базе RBM, позволяющий эффективно обучать сложные модели с использованием как безучителя, так и последующего дообучения с учителем.