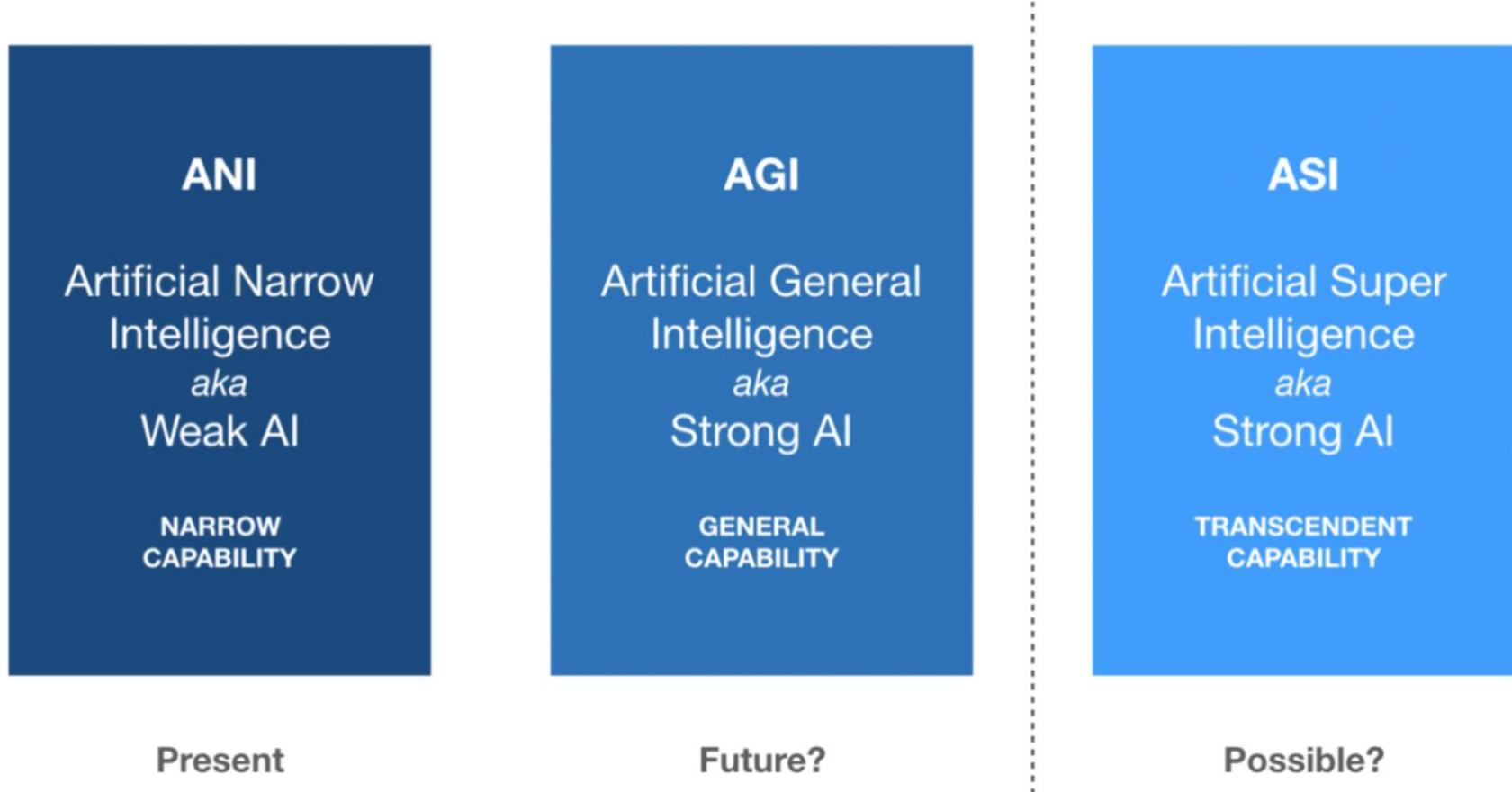


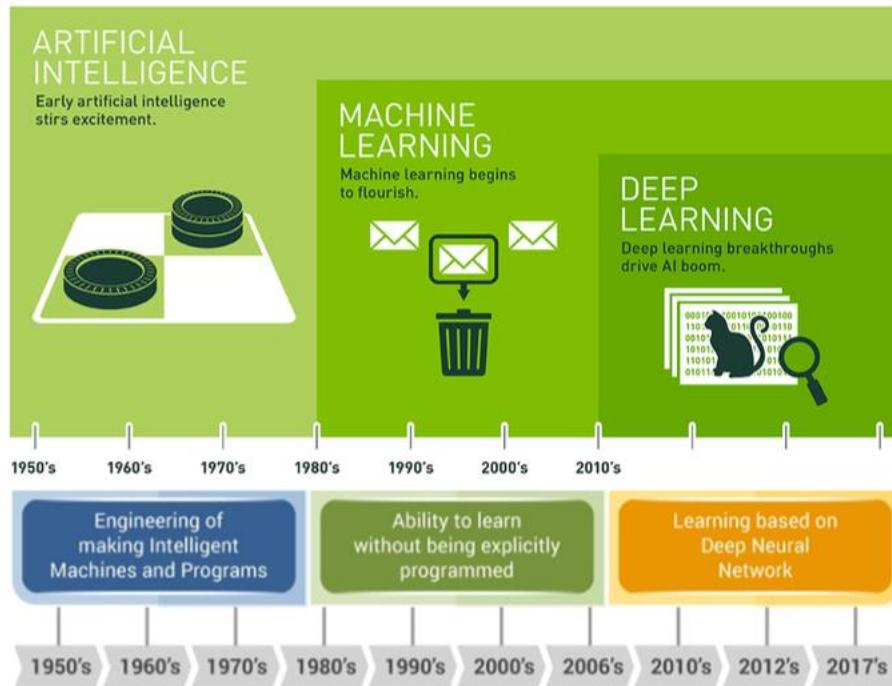
1. Что такое машинное обучение? Кто такой Data Scientist? Как машинное обучение и наука о данных связаны с искусственным интеллектом?
2. Уровни развития искусственного интеллекта (слабый, сильный, ANI, AGI, ASI).

Типы искусственного интеллекта



3. История развития ИИ, МО и глубокого обучения.

История развития ИИ, МО и глубокого обучения



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

Области применения машинного обучения



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 на примерах

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

Дано:

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

Найти:

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

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

Описание объектов. Векторы признаков

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

Типы скалярных признаков:

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

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

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

Описание ответов. Типы задач МО

Задача классификации:

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

Задача регрессии:

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

Задача ранжирования:

- Y – конечное отсортированное множество.

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

Статистическое (машинное) обучение с учителем

= обучение по прецедентам

= восстановление зависимости по эмпирическим данным

= предсказательное моделирование

= аппроксимация функций по заданным точкам

Два основных типа задач — **классификация и регрессия**

+ еще ранжирование

Алгоритм обучения

Процесс обучения с учителем состоит из двух этапов:

- **Обучение** (train):

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

- **Применение** (test):

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

9. Оценивание моделей. Эмпирический риск и функция потерь.

Функция потерь $\mathcal{L}(g, x)$: для заданного объекта $x \in X$ вычисляет величину ошибки алгоритма (функции) $g \in A$ на этом объекте. Ошибка тем больше, чем сильнее $g(x, \theta)$ отклоняется от правильного ответа $\varphi(x)$.

Функция потерь для задач классификации:

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

Функция потерь для задач регрессии:

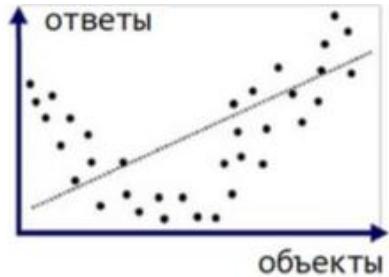
- $\mathcal{L}(g, x) = |g(x, \theta) - \varphi(x)|$ – абсолютное значение ошибки;
- $\mathcal{L}(g, x) = (g(x, \theta) - \varphi(x))^2$ – квадратичная ошибка.

Эмпирический риск

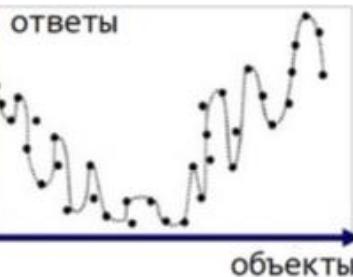
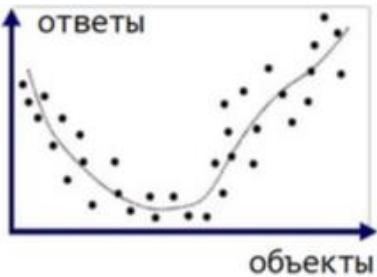
- Нельзя заранее достоверно узнать, на сколько хорошо алгоритм g покажет себя на практике («риск»), поскольку неизвестен истинный закон распределения данных $P(x, y)$.
- Оценить и улучшить работу алгоритма g можно на заранее известной ограниченной обучающей выборке (закон больших чисел).
- **Эмпирический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

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

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



недообучение



переобучение

- Недообучение (underfitting):
данных много,
параметров недостаточно,
модель простая, негибкая
- Переобучение (overfitting):
данных мало, параметров
слишком много, модель
сложная, избыточно гибкая



- Из-за чего возникает переобучение?
 - Избыточные параметры в модели $g(x, \theta)$ «расходуются» на чрезмерно тонкую подгонку под обучающую выборку.
 - Выбор g из A производится по неполной информации X^ℓ
- Как обнаружить переобучение?
 - Эмпирически, путем разбиения выборки на **train** и **test** (для **test** должны быть известны правильные ответы)
- Избавиться от переобучения нельзя. Как его минимизировать?
 - Увеличивать объем обучающих данных (big data).
 - Накладывать ограничения на θ (регуляризация).
 - Минимизировать одну из теоретических оценок.
 - Выбирать модель по оценкам обобщающей способности.

Многомерная линейная регрессия

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

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

Многомерная линейная регрессия

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

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

- Матричные обозначения:

$$\mathbf{F}_{\ell \times n} = \begin{pmatrix} f_1(x_1) & \cdots & f_n(x_1) \\ \vdots & \ddots & \vdots \\ f_1(x_\ell) & \cdots & f_n(x_\ell) \end{pmatrix}, \quad \mathbf{y}_{\ell \times 1} = \begin{pmatrix} y_1 \\ \vdots \\ y_\ell \end{pmatrix}, \quad \theta_{n \times 1} = \begin{pmatrix} \theta_1 \\ \vdots \\ \theta_n \end{pmatrix}$$

- Функционал квадрата ошибки:

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

12. Конструирование признаков.

- Использование интуиции для создания новых признаков путем преобразования или комбинирования оригинальных признаков.
 - Пример: предсказание стоимости жилья.
Признаки: x_1 – площадь квартиры (м. кв.), x_2 – город (категориальный). Модель:
$$g_1(x, \theta) = \theta_2 x_2 + \theta_1 x_1 + \theta_0$$
Добавляем новый признак: $x_3 = x_1 x_2$, чтобы напрямую учесть в модели различия стоимости кв. метра в разных регионах.
$$g_2(x, \theta) = \theta_3 x_3 + \theta_2 x_2 + \theta_1 x_1 + \theta_0$$
- Способы конструирования признаков
 - Полиномиальные признаки: возведение существующих признаков в степень или их комбинирование
 - Агрегация данных: среднее, сумма или медиана по имеющимся признакам
 - Лаги – значения за предыдущие периоды, которые могут влиять на текущие
 - Временные признаки – день недели, месяц или номер квартала, которые учитывают сезонные (периодические) изменения в данных
 - Знания из предметной области

13. Нормализация признаков.

- Нормирование значений признаков (нормализация средним):

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

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

- Минимаксная нормализация:

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

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

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

Когда нужна нормализация данных

- Необходимо стремиться: $-1 \leq x_j \leq 1$ для каждого признака x_j
- Эвристика: нормализация не обязательна, если диапазон признака отличается от $-1 \leq x_j \leq 1$ менее, чем на 2 порядка
- Нормализация не нужна:
$$\begin{aligned} -3 \leq x_1 &\leq 3 \\ -0.3 \leq x_2 &\leq 0.3 \\ 0 \leq x_3 &\leq 3 \\ -2 \leq x_4 &\leq 0.5 \end{aligned}$$
- Нормализация обязательна:
$$\begin{aligned} -100 \leq x_5 &\leq 100 \\ -0.001 \leq x_6 &\leq 0.001 \\ 98.6 \leq x_7 &\leq 105 \end{aligned}$$
- Лучше провести нормализацию, чем от нее отказаться

14. Метод наименьших квадратов.
15. Алгоритм градиентного спуска.

Алгоритм градиентного спуска

Повторять:

$$\theta_j^{\text{след.}} := \theta_j^{\text{пред.}} - \alpha \frac{\partial}{\partial \theta_j} Q(\theta_0^{\text{пред.}}, \theta_1^{\text{пред.}}, \dots, \theta_n^{\text{пред.}})$$

... до тех пор, пока не выполнится $|\theta_j^{\text{след.}} - \theta_j^{\text{пред.}}| < \delta_\theta$ для всех j ,

либо $|Q(\theta_0^{\text{пред.}}, \dots, \theta_n^{\text{пред.}}) - Q(\theta_0^{\text{след.}}, \dots, \theta_n^{\text{след.}})| < \delta_Q$.

Важно: обновлять θ_j необходимо одновременно для всех j

α – скорость обучения (learning rate):

- Если α слишком мало, требуется большое количество итераций для сходимости
- Если α слишком велико, значение функции потерь может не уменьшаться на каждой итерации и алгоритм может не сойтись к устойчивому минимуму

- Найдем частные производные функционала качества:

$$\begin{aligned}\frac{\partial}{\partial \theta_0} Q(\theta_0, \theta_1) &= \frac{\partial}{\partial \theta_0} \frac{1}{2l} \sum_{i=1}^l (g(x_i, \theta_0, \theta_1) - y_i)^2 = \frac{1}{2l} \sum_{i=1}^l \frac{\partial}{\partial \theta_0} (\theta_1 x_i + \theta_0 - y_i)^2 = \\ &= \frac{1}{l} \sum_{i=1}^l (\theta_1 x_i + \theta_0 - y_i)\end{aligned}$$

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

- Итоговые формулы для градиентного спуска:

$$\theta_0 := \theta_0 - \alpha \frac{\partial}{\partial \theta_0} Q(\theta_0, \theta_1) = \theta_0 - \alpha \frac{1}{l} \sum_{i=1}^l (g(x_i, \theta_0, \theta_1) - y_i)$$

$$\theta_1 := \theta_1 - \alpha \frac{\partial}{\partial \theta_1} Q(\theta_0, \theta_1) = \theta_1 - \alpha \frac{1}{l} \sum_{i=1}^l (g(x_i, \theta_0, \theta_1) - y_i) x_i$$

16. Стохастический градиентный спуск.

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

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

1. $\theta_j = 0$ для всех $j = 0, \dots, n$.

2. Небольшие случайные значения:

$$\theta_j = \text{random}\left(-\frac{1}{2n}, \frac{1}{2n}\right)$$

3. $\theta_j = \frac{\langle y, f_j \rangle}{\langle f_j, f_j \rangle}$, $f_j = (f_j(x_i))_{i=1}^\ell$ – вектор значений признака.

Эта оценка θ оптимальна, если:

- 1) функция потерь квадратична и
- 2) признаки некоррелированы, $\langle f_j, f_k \rangle = 0$, $j \neq k$.

4. Обучение по небольшой случайной подвыборке объектов.
5. Мультистарт: многократные запуски из разных случайных начальных приближений и выбор лучшего решения.

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

- Штраф за увеличение нормы вектора весов:

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

- Градиент:

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

- Модификация градиентного шага:

$$\theta = \theta(1 - \alpha \tau) - \alpha \nabla \mathcal{L}_i(\theta).$$

- Методы подбора коэффициента регуляризации τ :

1. скользящий контроль;
2. стохастическая адаптация;
3. двухуровневый байесовский вывод.

19. Повышение производительности с помощью векторизации.

Векторизация

- Регрессионная модель:

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

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

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

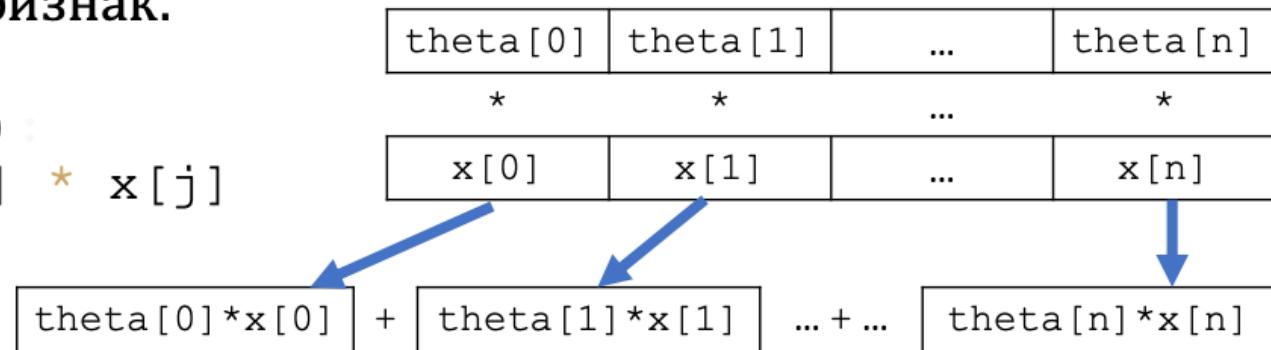


- В векторном виде:

$$g(\mathbf{x}_i, \theta) = \theta \cdot \mathbf{x}_i$$

быстрее на порядки!

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



20. Постановка задачи бинарной и многоклассовой классификации.

Задача обучения классификации

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

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

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

- Функция потерь – бинарная или ее **аппроксимация**:

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

- Метод обучения – *минимизация эмпирического риска*:

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

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

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

Задача многоклассовой классификации

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

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

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

- Функция потерь – бинарная или ее аппроксимация:

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

- Метод обучения – минимизация эмпирического риска:

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

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

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

Разделяющие классификаторы (margin-based classifier)

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

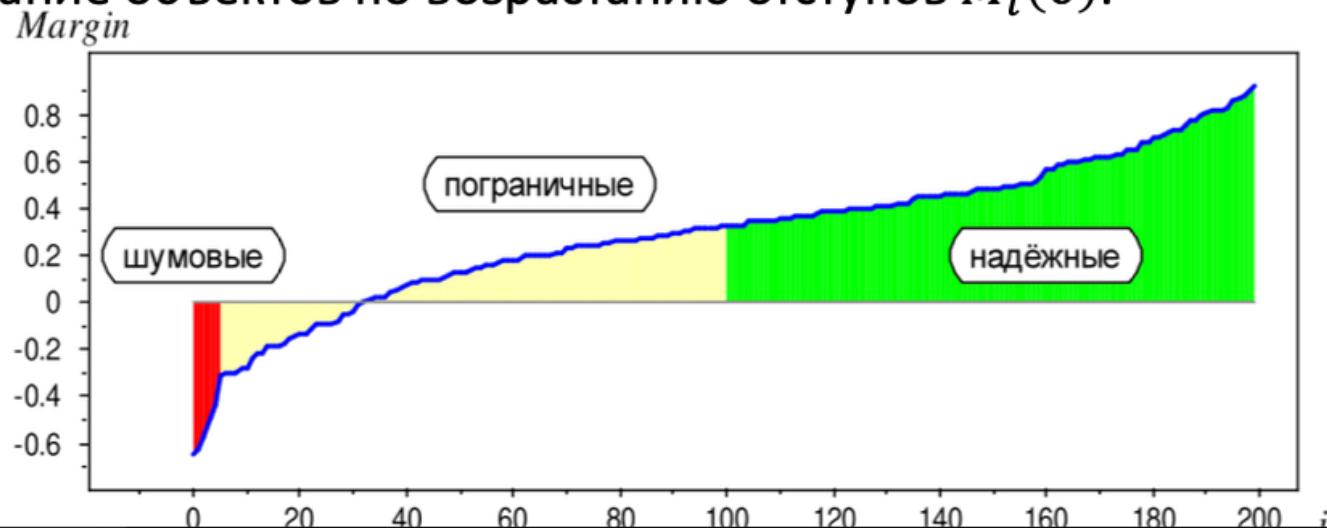
$h(x, \theta)$ – разделяющая (дискриминантная) функция

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

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

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

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



Многоклассовый классификатор:

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

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

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

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

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

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

22. Логистическая регрессия.

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:

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

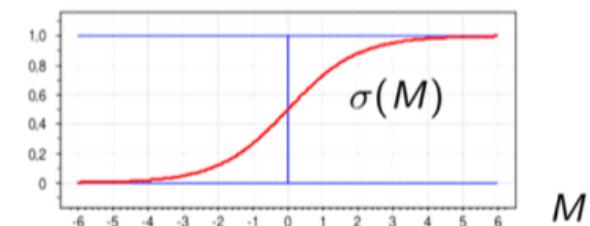
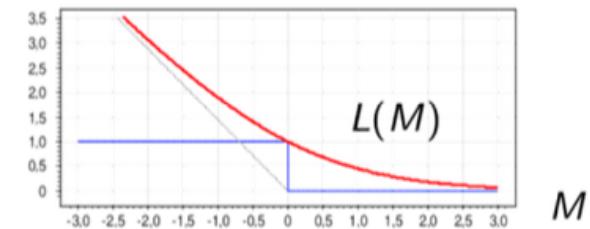
Модель условной вероятности:

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

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

Максимизация правдоподобия (logistic loss) с регуляризацией:

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



Многоклассовая 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:

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

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 $p(x, y)$

$p(x, y) = P(y|x; \Theta)p(x)$ – параметризация плотности:

$P(y|x; \Theta)$ – условная вероятность класса y ;

$p(x)$ – 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.

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

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

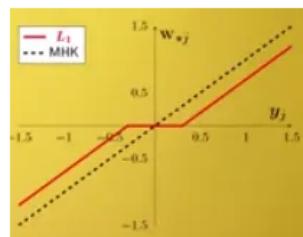
$$L_1 = \sum_i (y_i - y(t_i))^2 + \lambda \sum_i |a_i|.$$

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.



<https://www.coursera.org/lecture/supervised-learning/rieghuliarizatsiia-sR94Q>

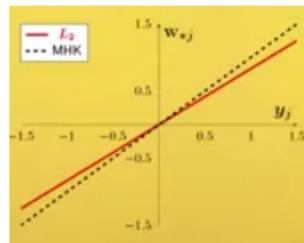
L2-регуляризация, или регуляризация Тихонова (в англоязычной литературе – ridge regression или Tikhonov regularization), для интегральных уравнений позволяет балансировать между соответствием данным и маленькой нормой решения:

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

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.



<https://www.coursera.org/lecture/supervised-learning/rieghuliarizatsiia-sR94Q>

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

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

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

- Метод парзеновского окна *фиксированной ширины*:

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

- Метод парзеновского окна *переменной ширины*:

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

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

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

Разделяющая полоса (разделяющая гиперплоскость посередине):

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

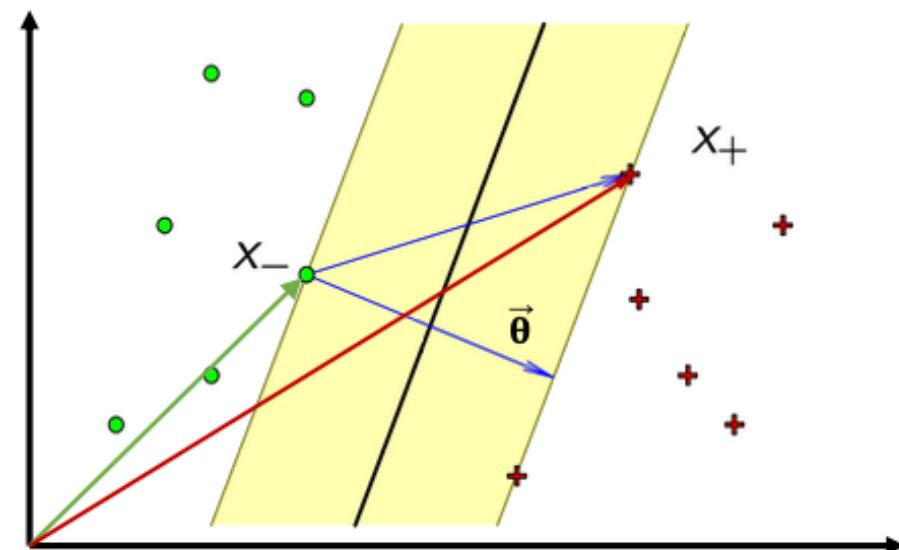
$$\exists \vec{x}_+: \langle \vec{x}_+, \vec{\theta} \rangle - \theta_0 = +1$$

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

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

Ширина полосы:

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



Геометрическая интерпретация

Дано:

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

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

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

Найти:

класс, к которому относится \vec{x} .

Решение:

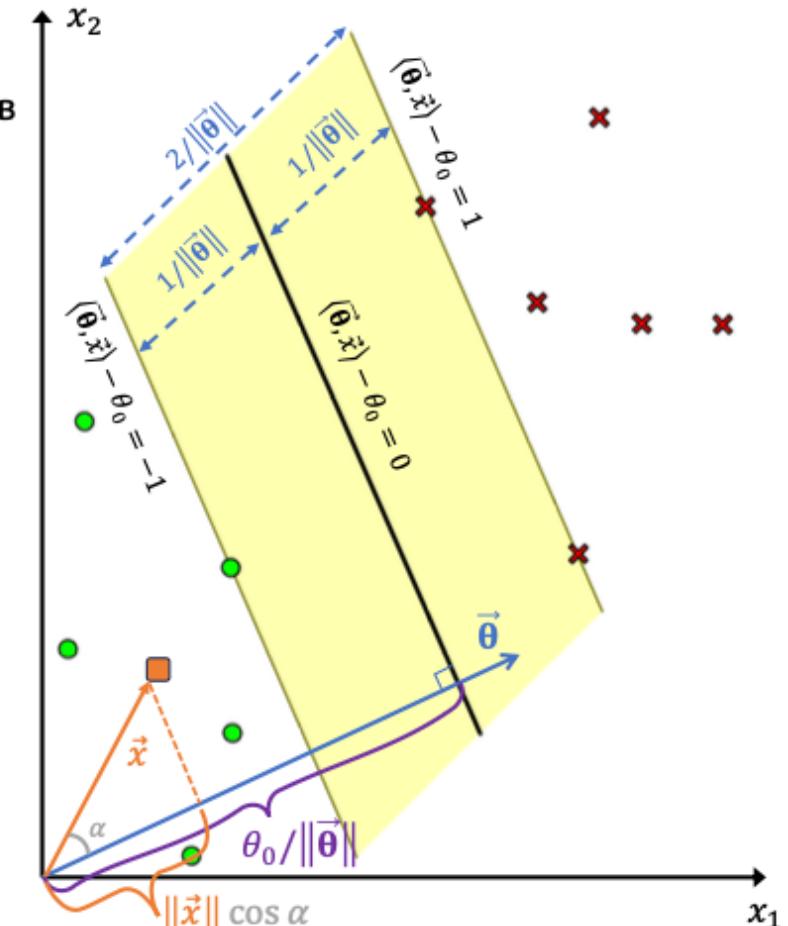
Чтобы узнать, по какую сторону от разделяющей гиперплоскости находится объект \vec{x} , достаточно спроектировать вектор \vec{x} на вектор $\vec{\theta}$ и сравнить длину получившейся проекции с пороговым значением θ_0 :

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

Решающее правило:

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

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



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 , $i = 1, \dots, \ell$:

1. $\lambda_i = 0; \eta_i = C; \xi_i = 0; M_i \geq 1$ – периферийный.
2. $0 < \lambda_i < C; 0 < \eta_i < C; \xi_i = 0; M_i = 1$ – **опорный**-границный.
3. $\lambda_i = C; \eta_i = 0; \xi_i > 0; M_i < 1$ – **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)

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

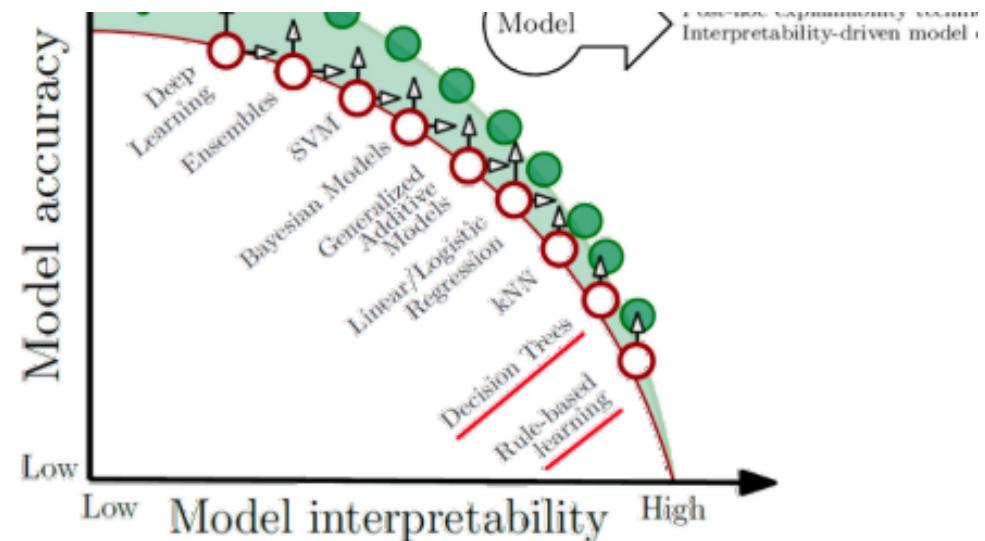
32. Интерпретируемость алгоритмов машинного обучения.

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, $\forall v \in V_{\text{внутр}}$;

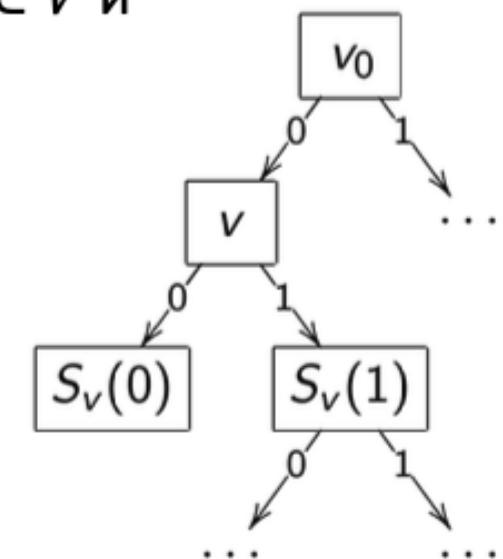
$S_v: D_v \rightarrow V$ – множество дочерних вершин;

$y_v \in Y$ – 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.

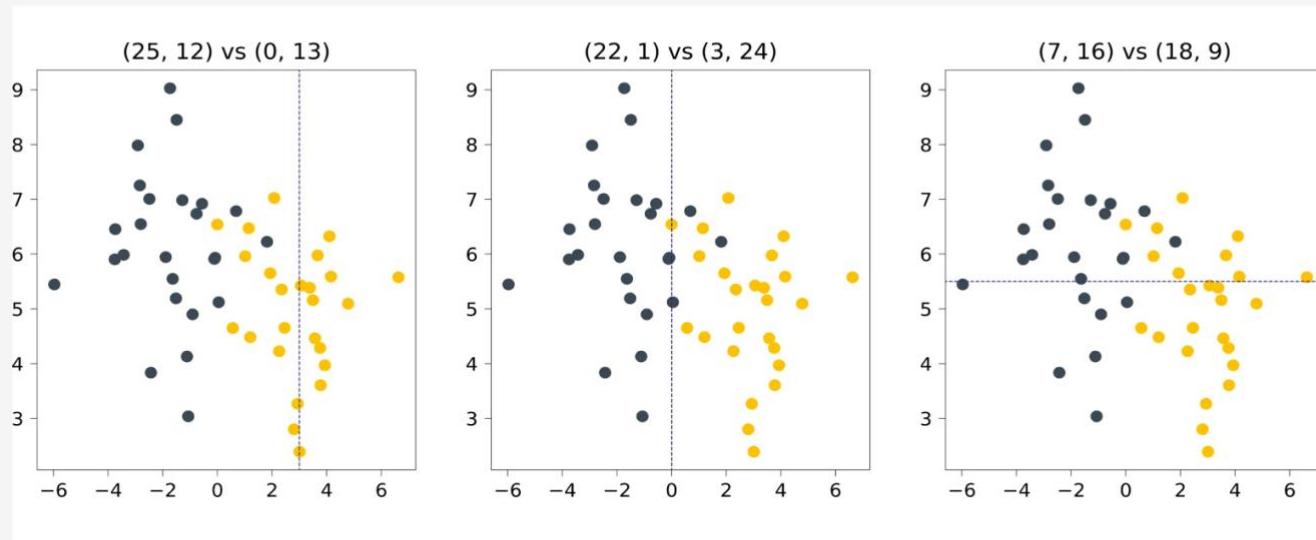
Так как $p_k \in [0, 1]$, энтропия неотрицательна. Если случайная величина принимает только одно значение, то она абсолютно предсказуема и её энтропия равна -1
 $\log(1) = 0$.

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

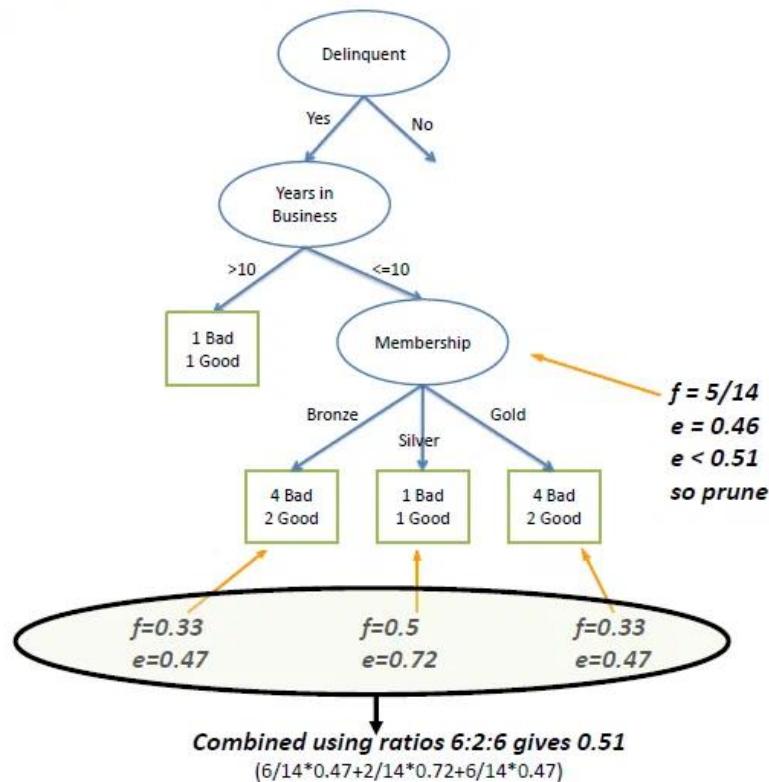
Error estimate for a sub-tree is weighted sum of error estimates for all its leaves. The error estimate (e) for a node is:

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

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

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

39. Метрики оценки качества регрессии.

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

Постановка задачи кластеризации

Дано:

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

Найти:

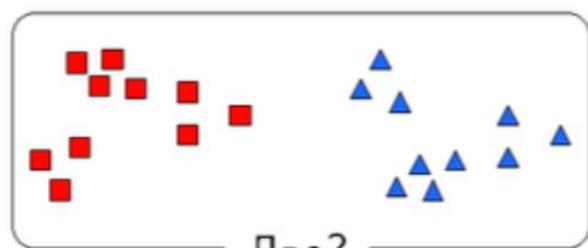
- Y – множество кластеров,
- $g: X \rightarrow Y$ – алгоритм кластеризации, такой что:
 - каждый кластер состоит из близких объектов;
 - Объекты разных кластеров существенно различны.

Это задача *обучения без учителя* (unsupervised learning).

Решение задачи кластеризации принципиально неоднозначно:

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

Пример: сколько здесь кластеров?



Два?



Четыре?



Шесть?

- 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:

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

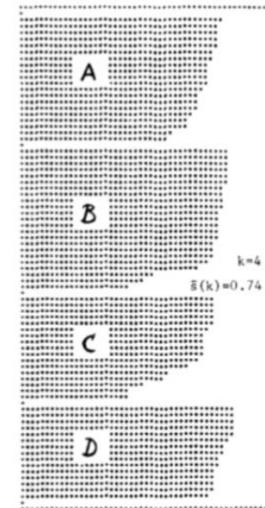
- Отношение пары функционалов: $\Phi_0 / \Phi_1 \rightarrow \min$.

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:

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

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

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

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:

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

Алгоритм Ллойда

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

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

повторять

отнести каждый x_i 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^ℓ , $K = |Y|$;

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

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

повторять

отнести каждый $x_i \in U$ к ближайшему центру:

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

вычислить новые положения центров:

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

пока a_i не перестанут изменяться;

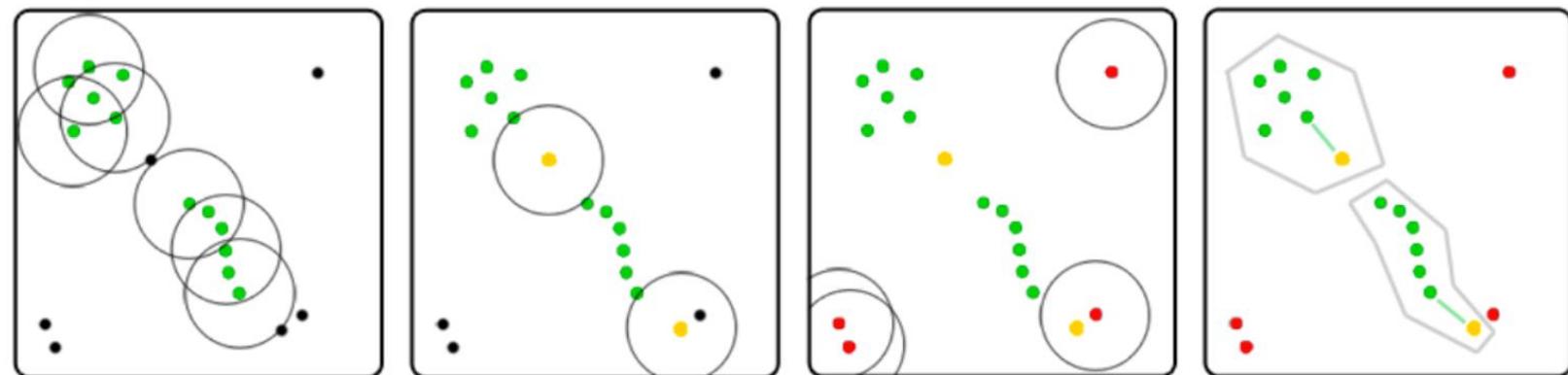
44. Алгоритм DBSCAN.

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

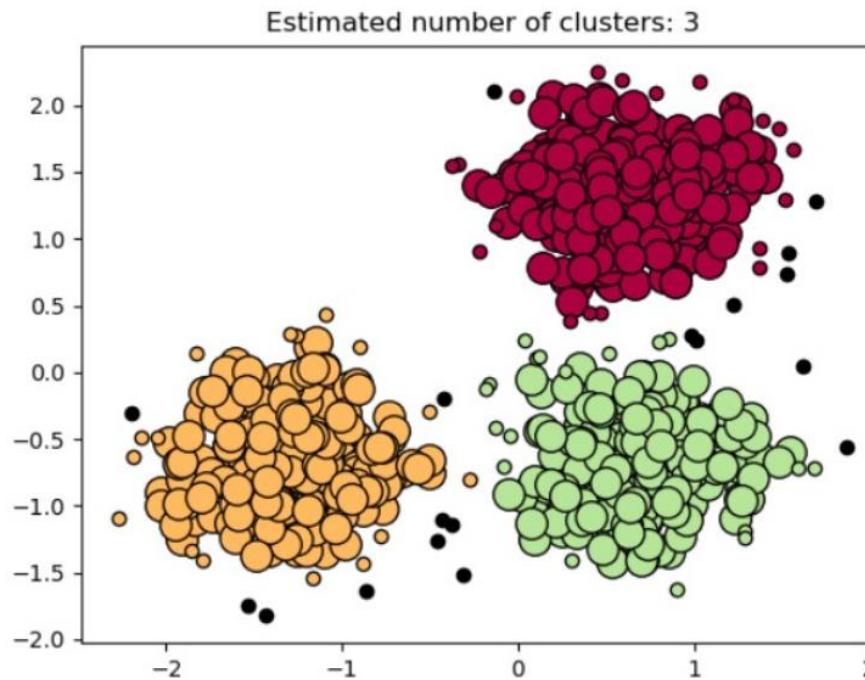
если $|U_\varepsilon(x')| \geq m$ **то** $K := K \cup U_\varepsilon(x')$;

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

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

R^C не монотонно; R^B , R^A , R^Γ , R^Y — монотонны.

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

46. Карты Кохонена.

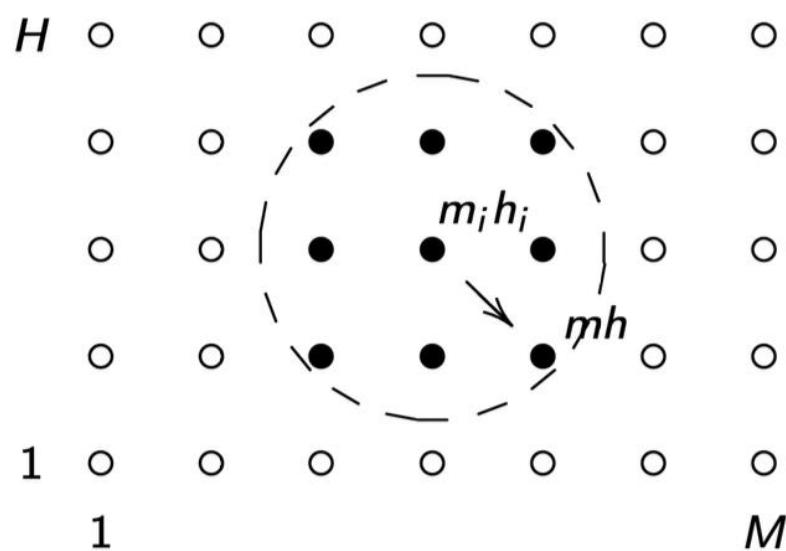
$Y = \{1, \dots, M\} \times \{1, \dots, H\}$ — прямоугольная сетка кластеров

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 $M \times H$:

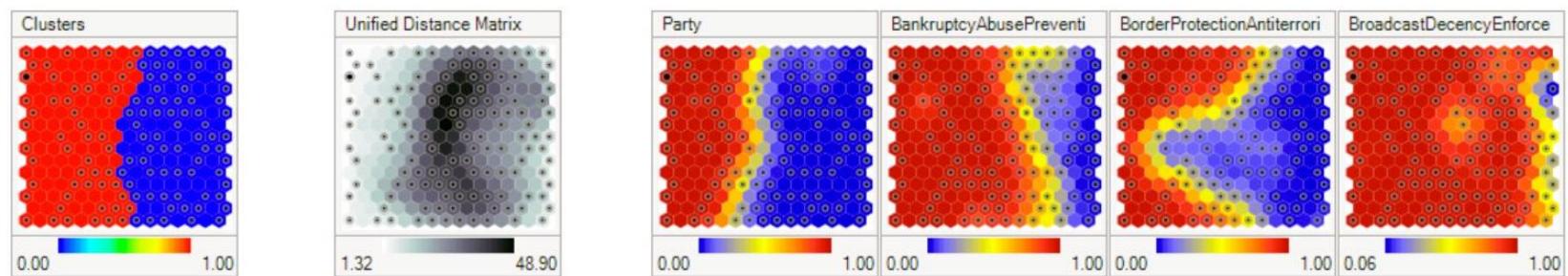
- Цвет узла (m, h) — локальная плотность в точке (m, h) — среднее расстояние до k ближайших точек выборки
- По одной карте на каждый признак:
цвет узла (m, h) — значение j -й компоненты вектора $w_{m,h}$

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.

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 ансемблирования: 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 $1 - \left(1 - \frac{1}{\ell}\right)^\ell \rightarrow 1 - \frac{1}{e} \approx 63.2\%$ 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;

bagging

+

random subspaces

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;

T

49. Случайный лес.

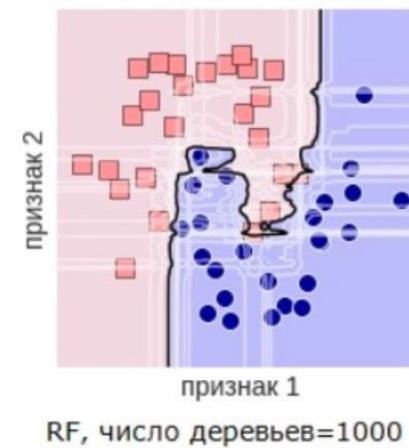
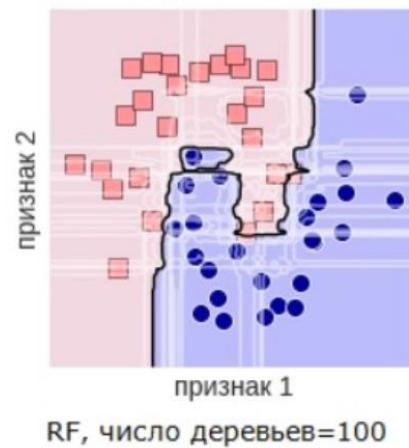
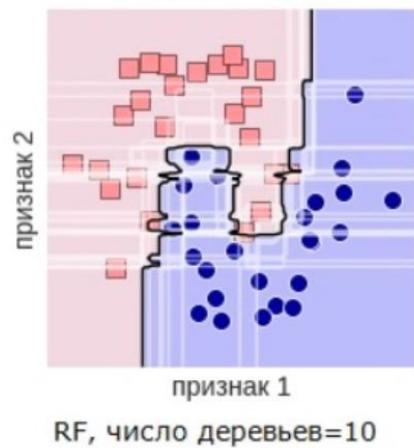
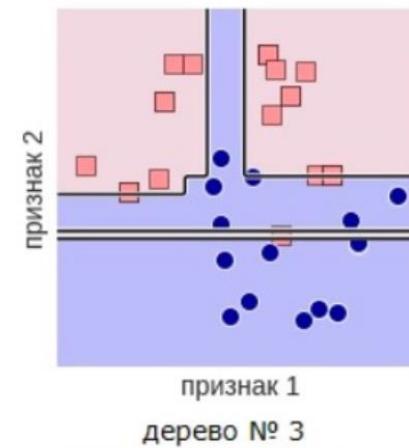
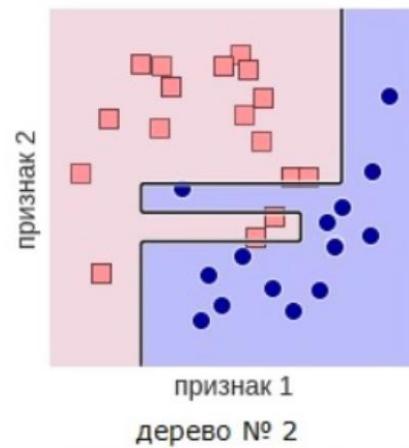
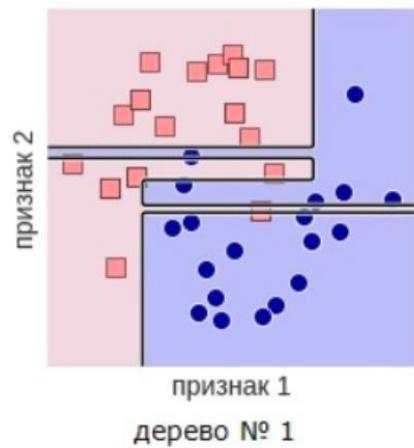
Обучение случайного леса:

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

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

- 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.

Идея метода

Бэггинг – технология классификации, где в отличие от [бустинга](#) все элементарные классификаторы обучаются и работают параллельно (независимо друг от друга). Идея заключается в том, что классификаторы не исправляют ошибки друг друга, а компенсируют их при голосовании. Базовые классификаторы должны быть независимыми, это могут быть классификаторы основанные на разных группах методов или же обученные на независимых наборах данных. Во втором случае можно использовать один и тот же метод.

Бэггинг на подпространствах

Этот алгоритм применяется для классификации многомерных объектов. Рассматриваемый алгоритм помогает добиться качественной классификации в условиях, когда разделить объекты на группы на всем пространстве параметров не представляется возможным. Предлагается разделить пространство характеристик на подмножества объединенных по смыслу параметров. Классификация на каждом подпространстве производится отдельно, затем результаты учитываются в голосовании. В этом случае будет учтен вклад каждой смысловой группы и много повысится вероятность того, что итоговые результаты классификации окажутся более качественными нежели без деления на подпространства, так как параметры, по которым представители разных классов неотличимы, попадут, почти наверняка, не во все группы.

Бустинг

Бустинг (англ. boosting — улучшение) — это процедура последовательного построения [композиции алгоритмов машинного обучения](#), когда каждый следующий алгоритм стремится компенсировать недостатки композиции всех предыдущих алгоритмов. Бустинг представляет собой жадный алгоритм построения [композиции алгоритмов](#). Изначально понятие бустинга возникло в работах по [вероятно почти корректному обучению](#) в связи с вопросом: возможно ли, имея множество плохих (незначительно отличающихся от случайных) алгоритмов обучения, получить хороший^[1].

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:

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

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

52. Градиентный бустинг.

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:

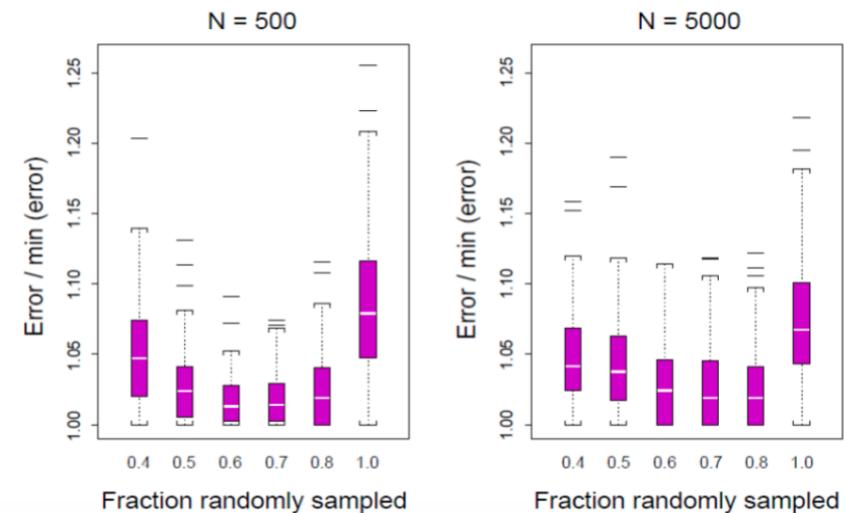
- улучшается сходимость, уменьшается время обучения
- улучшается обобщающая способность ансамбля
- можно использовать несмешённые оценки out-of-bag

Эксперименты:

относительная ошибка при различном объёме выборки N

Вывод:

оптимально сэмплировать около 60–80% выборки



Основные мотивации CatBoost

Две проблемы:

- Надо обрабатывать категориальные признаки с большим числом редких значений (пользователь, регион, город, реклама, рекламодатель, товар, документ, автор, и т.д.)
- Переобучение (смещённость, target leakage) в градиентах: $g_i = \mathcal{L}'(a_{t-1}(x_i), y_i)$ вычисляются в тех же точках x_i , по которым ансамбль $a_{t-1}(x)$ обучался аппроксимировать y_i

Приём, похожий на Out-Of-Bag и на онлайновые методы:

- для получения несмещённых оценок на объекте x ; хранить и дообучать ансамбль на выборках без этого объекта
- как сделать, чтобы этих выборок было $O(\log \ell)$, а не $O(\ell)$?
- как сделать, чтобы они не сильно перекрывались?

Основные особенности CatBoost:

- Работа с категориальными признаками. CatBoost уникален благодаря способности обрабатывать категориальные данные без необходимости в предварительном кодировании. Это позволяет значительно упростить подготовку данных и сохранить информацию, которая может быть утрачена при использовании методов вроде one-hot encoding.
- Обработка пропусков. Алгоритм автоматически обрабатывает пропущенные значения, что уменьшает вероятность возникновения ошибок, связанных с отсутствием данных.
- Борьба с переобучением. CatBoost использует несколько методов для предотвращения переобучения, включая мощную регуляцию и усреднение.
- Скорость и производительность. В CatBoost реализованы оптимизации, делающие обучение и предсказания более быстрыми по сравнению с аналогами. Они включают в себя поддержку многоядерной обработки и эффективное использование памяти.
- Стабильность и воспроизводимость. Алгоритм предлагает методы, обеспечивающие стабильные результаты даже при изменении порядка входных данных.

CatBoost реализует улучшенный подход к градиентному бустингу, который включает в себя следующие этапы:

- Инициализация. Алгоритм начинается с простой модели, которая предсказывает среднее значение целевой переменной.
- Построение деревьев. На каждом шаге создаётся новое дерево решений, обучающееся на

- Использование случайных перестановок. Для борьбы с переобучением CatBoost применяет метод случайных перестановок данных, что помогает улучшить обобщающую способность модели.
- Вычисление градиента. На каждой итерации вычисляется градиент ошибки, который затем используется для корректировки модели.
- Интеграция модели. Итоговое предсказание — это взвешенная сумма всех деревьев, где каждый последующий шаг уменьшает ошибку модели.

CatBoost выделяется среди других алгоритмов бустинга способностью эффективно работать с категориальными данными и предоставлять точные предсказания, минимизируя риск переобучения и обеспечивая высокую скорость обучения.

В XGBoost ответы суммируются по всем деревьям ансамбля:

$$F(x) = \sum_{k=1}^K f_k(x)$$

Суммарная функция потерь в XGBoost выглядит следующим образом:

$$\text{total_loss} = \sum_{i=1}^N \text{loss}(y_i, F(x_i)) + \gamma \sum_{k=1}^K T_k + \frac{1}{2} \lambda \sum_{k=1}^K \|w_k\|^2$$

Здесь γ , λ - гиперпараметры. Первое слагаемое - это основная функция потерь, второе слагаемое штрафует деревья за слишком большое количество листьев, третье слагаемое за слишком большие предсказания. Третье слагаемое является нетипичным в машинном обучении. Оно обеспечивает то, что каждое дерево вносит минимальный вклад в результат. Функция потерь используется при построении каждого следующего дерева, то есть функция потерь оптимизируется по параметрам лишь последнего дерева, не затрагивая предыдущие. Для минимизации *total_loss* используется метод второго порядка, то есть рассчитываются не только производные, но и вторые производные функции потерь по предсказаниям предыдущих деревьев (подробнее см. [Chen and Guestrin, 2016](#), раздел 2.2).

При поиске каждого нового разделяющего правила, для каждого признака перебираются не все возможные значения порога, а значения с определенным шагом. Для этого на признаке рассчитывается набор персентилей, используя статистику из обучающего датасета. Поиск оптимального порога выполняется только среди этих персентилей. Это позволяет существенно сократить время перебора и ускорить обучение. Для работы с пропущенными значениями в каждом решающем правиле определяется ветвь, в которую будут отправлены объекты с пропущенным значением данного признака.

Однако основной ценностью библиотеки XGBoost является эффективная программная реализация. За счет разных оптимизаций, таких как эффективная работа с пропущенными значениями, поиск порога только среди персентилей, оптимизация работы с кэшем и распределенное обучение, достигается выигрыш в десятки или даже сотни раз по сравнению с native реализацией.

54. Работа с большими данными. Экосистема Apache Hadoop.

- Вертикальное масштабирование
- Отправка кода к данным
- Отказоустойчивость
- Горизонтальное масштабирование
- Перемещение данных к коду
- Инкапсуляция сложности реализации распределенной системы

Какие функции выполняют вендоры дистрибутивов?

Выберите все подходящие ответы из списка

 Всё получилось!

- Разрабатывают дополнительные фичи в стандартных компонентах Hadoop
- Предоставляют дистрибутивы в различных форматах (rpm, tar.gz, образ виртуальных машин)
- Исправляют ошибки в компонентах Hadoop
- Обеспечивают совместимость разных компонентов Hadoop

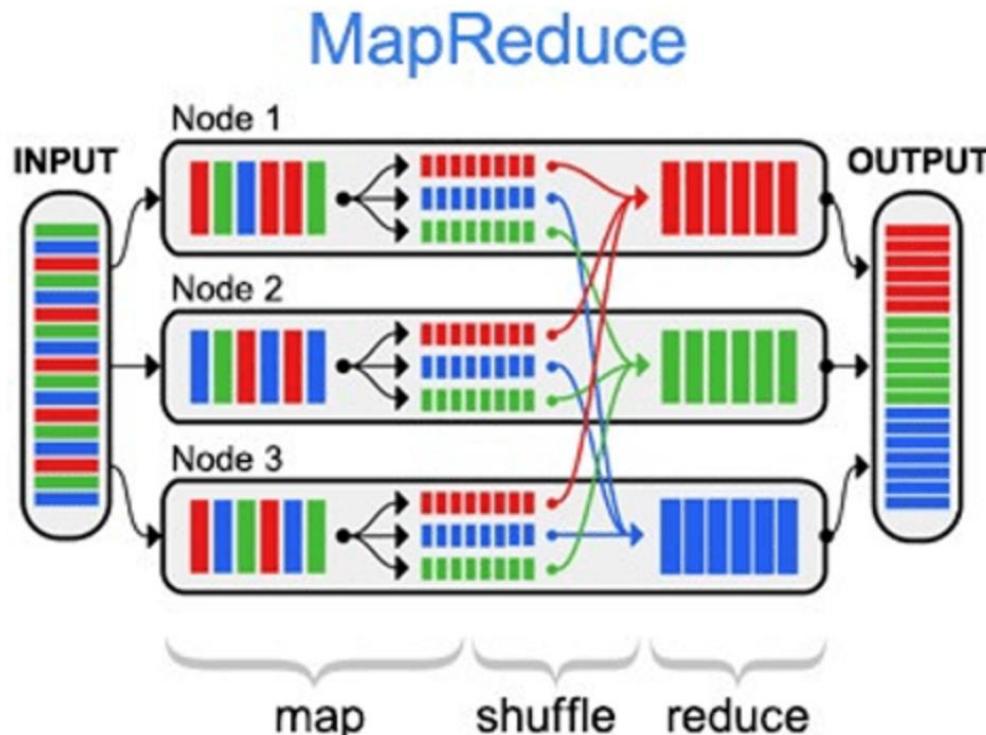
55. Файловая система HDFS.

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 (реплицированными) 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 (>64MB), поскольку HDFS предназначена для хранения большого количества огромных (>10GB) 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 объём 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

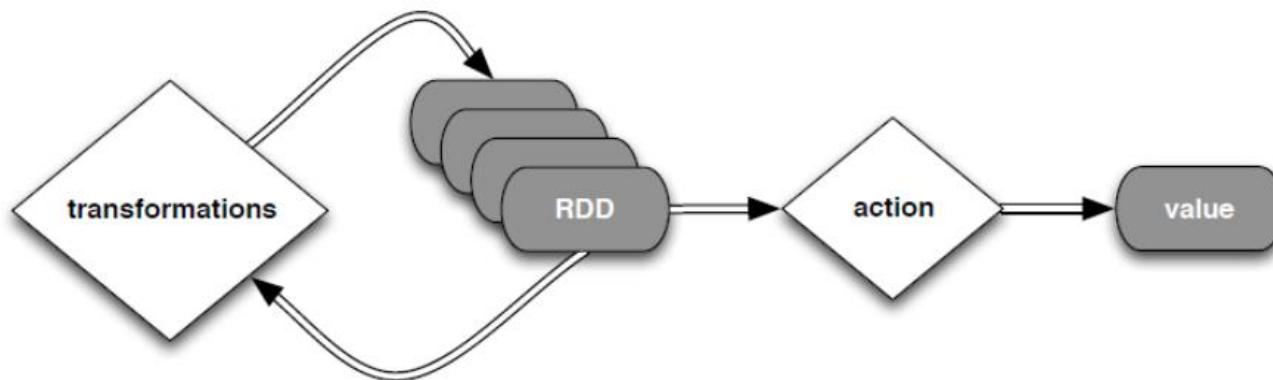
Два типа наборов данных:

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

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

58. Принципы работы рекомендательных систем.

Обозначения

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} = [пользователь u посетил/лайкнул документ i]

n_{iw} = частота слова w в документе i

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

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

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

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

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

59. Коллаборативная и контентная фильтрация.

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

Резюме по Memory-Based методам

Преимущества для бизнес-приложений:

- легко понимать и объяснять:
«те, кто купил эту книгу, также покупали...» [Amazon.com]
- легко реализовать
- метод SLIM — простой и до сих пор один из лучших

Недостатки:

- требуется хранение огромной разреженной матрицы R
- проблема «холодного старта»
 - не ясно, что рекомендовать новым клиентам
 - не ясно, как и кому рекомендовать новые объекты
- иногда рекомендации тривиальны
 - предлагается всё наиболее популярное

Основные подходы в коллаборативной фильтрации (CF)

По используемой математической технике:

- *корреляционные модели* (Memory-Based CF):
корреляции строк/столбцов исходной матрицы R
- *латентные семантические модели* (Latent Model-Based CF):
векторные представления (эмбеддинги) клиентов и объектов

По типам используемых данных:

- 2D: только матрица R
- 3D: контекстно-зависимые (context-aware, field-aware)
- учёт содержимого (content-aware, гибридные модели)
- учёт связей доверия между клиентами (trust-aware)
- учёт времени (time-aware)

Основанные на контенте (content-based)

Этот тип лежит в основе многих рекомендательных систем. В отличие от коллаборативной фильтрации, этап знакомства с пользователем опускается. Товары и услуги рекомендуются на основе знаний о них: жанр, производитель, конкретные функции и т.п. В общем, применяют любые данные, которые можно собрать.

По такому принципу работают системы интернет-магазинов, онлайн-кинотеатров и других сервисов. Например, IMI выстраивает рекомендации по жанрам, странам-производителям фильмов, актерам и т.п.

Создатели платформ используют этот тип систем, чтобы не потерять новых пользователей, данных о которых еще нет. Отсюда же вытекают два недостатка: первое время системы действуют неточно и требуется больше времени на реализацию.

Основанные на знаниях (knowledge-based)

Этот тип работает на основе знаний о какой-то предметной области: о пользователях, товарах и других, которые могут помочь в ранжировании. Как и в случае с «content-based», оценки других пользователей системы не учитывают. Есть несколько разновидностей: case-based, demographic-based, utility-based, critique-based, whatever-you-want-based и т.д.

На самом деле количество подтипов ограничено фантазией создателей. При реализации нового проекта в зависимости от сферы деятельности в рекомендательную систему можно заложить любую предметную область и ранжировать по ней.

Например, магазин техники Apple «reStore» подбирает потенциальным покупателям наборы, в зависимости от просматриваемого товара:

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:

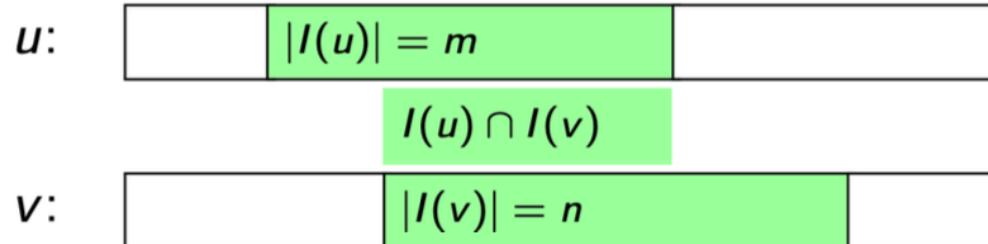
$$S(u, v) = \frac{\sum_{i \in I(u, v)} r_{ui} r_{vi}}{\sqrt{\sum_{i \in I(u)} r_{ui}^2} \sqrt{\sum_{i \in I(v)} r_{vi}^2}},$$

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:
 - жётская: $\begin{cases} p_{tu} = [\text{клиент } u \text{ принадлежит кластеру } t \in G]; \\ q_{ti} = [\text{объект } i \text{ принадлежит кластеру } t \in H]; \end{cases}$
 - 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:

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

65. Понятие искусственного нейрона.

Измерение качества рекомендаций

Многокритериальность в рекомендательных системах:

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

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

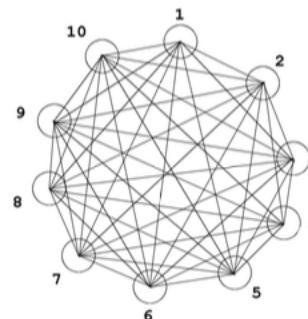
66. Сеть Хопфилда.

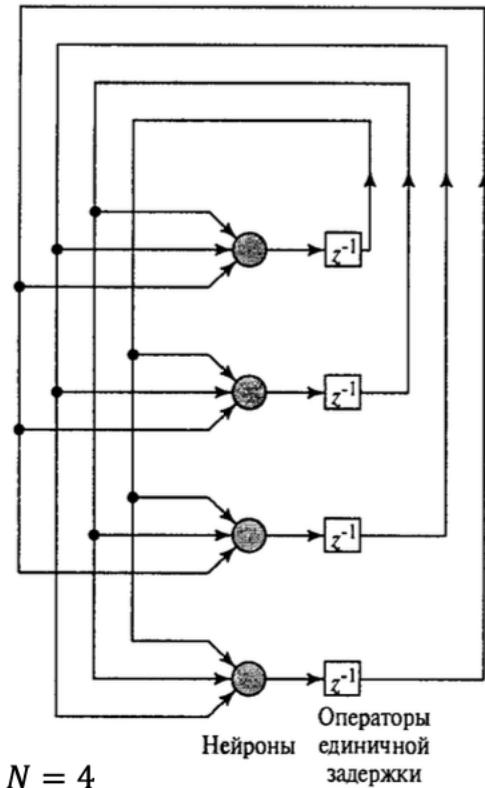
- Один из видов рекуррентных сетей
 - Полностью рекуррентный
 - Веса симметричны
 - Узлы могут быть только в состояниях *вкл* или *выкл*
 - Случайный порядок перехода от одного узла к другому

- Обучение: **Правило Хебба** (cells that fire together wire together)

- Может вспомнить и восстановить образ по его искаженной или неполной версии

→ ассоциативная память





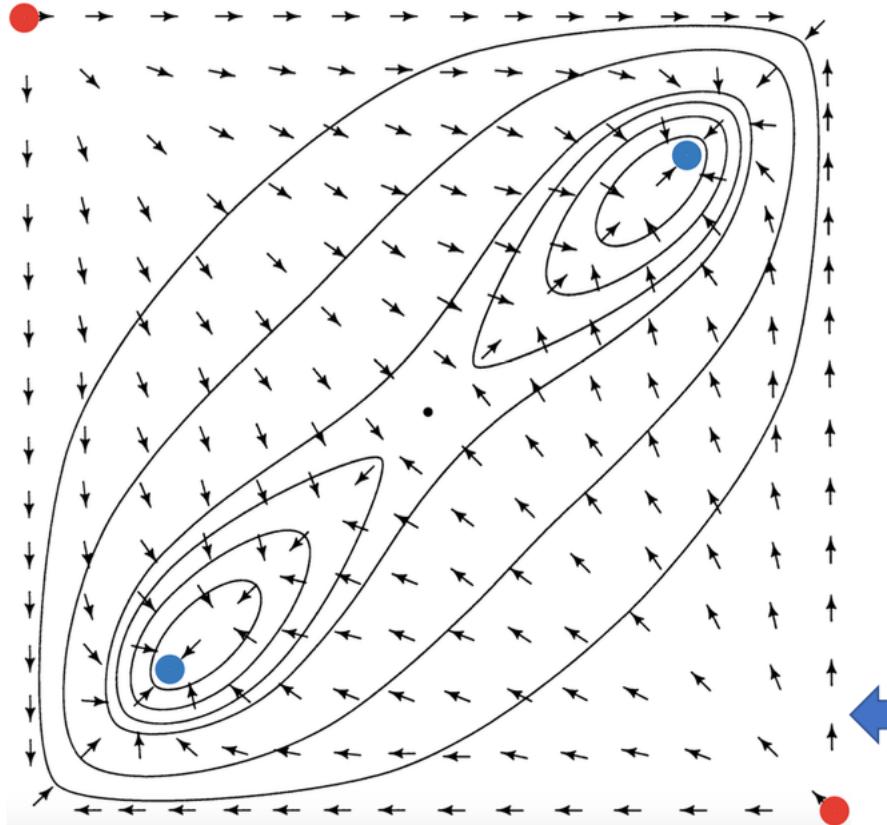
- Состояние сети (модели) из N нейронов:
 $\mathbf{y} = (y_1, y_2, \dots, y_N)^T$
- Дискретная сеть Хопфилда:

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

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

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 .

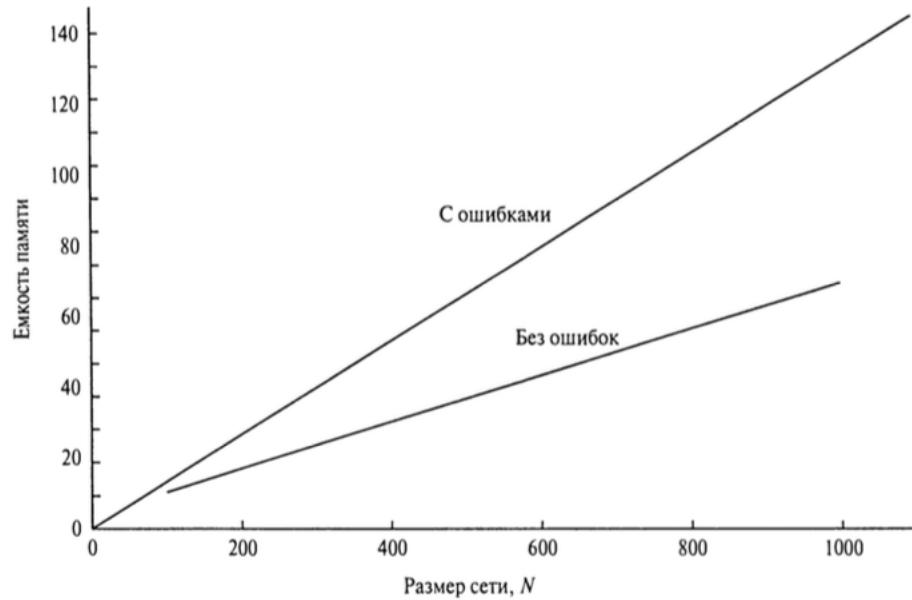
Контурная карта энергии непрерывной модели Хопфилда с двумя нейронами и двумя устойчивыми состояниями. Ординаты и абсциссы являются выходами двух нейронов. Устойчивые состояния соответствуют минимумам энергии и показаны синим, неустойчивые экстремумы – красным.

- 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 \mathbf{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:

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

67. Понятие стохастического нейрона. Машина Больцмана.

68. Алгоритм имитации отжига.

Алгоритм имитации отжига (англ. *simulated annealing*) — эвристический алгоритм глобальной оптимизации, особенно эффективный при решении дискретных и комбинаторных задач.

Алгоритм вдохновлён процессом [отжига](#) в металлургии — техники, заключающейся в нагревании и контролируемом охлаждении металла, чтобы увеличить его кристаллизованность и уменьшить дефекты. Симулирование отжига в переборных задачах может быть использовано для приближённого нахождения глобального минимума функций с большим количеством свободных переменных.

Для примера будем рассматривать задачу коммивояжёра:

Есть n городов, соединённых между собой дорогами. Необходимо проложить между ними кратчайший замкнутый маршрут, проходящий через каждый город только один раз.

Пусть имеется некоторая функция $f(x)$ от состояния x , которую мы хотим минимизировать. В данном случае x это перестановка вершин (городов) в том порядке, в котором мы будем их посещать, а $f(x)$ это длина соответствующего пути.

Возьмём в качестве базового решения какое-то состояние x_0 (например, случайную перестановку) и будем пытаться его улучшать.

Введём температуру t — какое-то действительное число (изначально равное единице), которое будет изменяться в течение оптимизации и влиять на вероятность перейти в соседнее состояние.

Пока не придём к оптимальному решению или пока не закончится время, будем повторять следующие шаги:

1. Уменьшим температуру $t_k = T(t_{k-1})$.
2. Выберем случайного соседа x — то есть какое-то состояние y , которое может быть получено из x каким-то минимальным изменением.
3. С вероятностью $p(f(x), f(y), t_k)$ сделаем присвоение $x \leftarrow y$.

В каждом шаге есть много свободы при реализации. Основные эвристические соображения следующие:

1. В начале оптимизации наше решение и так плохое, и мы можем позволить себе высокую температуру и риск перейти в состояние хуже. В конце наоборот — наше решение почти оптимальное, и мы не хотим терять прогресс. Температура должна быть высокой в начале и медленно уменьшаться к концу.
2. Алгоритм будет работать лучше, если функция $f(x)$ «гладкая» относительно этого изменения, то есть

быть больше при высокой температуре.

Например, можно действовать так:

1. $t_k = \gamma \cdot t_{k-1}$, где γ это какое-то число, близкое к единице (например, 0.99). Оно должно зависеть от планируемого количества итераций: оптимизация при низкой температуре почти ничего не будет менять.
2. В случае с перестановками этим минимальным изменением может быть, например, swap двух случайных элементов.
3. Если y не хуже, то есть $f(y) \leq f(x)$, то переходим в него в любом случае. Иначе делаем переход в y , с вероятностью $p = e^{\frac{f(x)-f(y)}{t_k}}$ — это экспонента отрицательного числа, и она даст вероятность в промежутке $(0, 1)$.

Вообще, в выборе конкретных эвристик не существует «золотого правила». Все компоненты алгоритма сильно зависят друг от друга и от задачи.

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

70. Глубокие сети на основе машины Больцмана.