Билеты по машинному обучению

ЭФ МГУ

2020-2021

Содержание

1 Матричное дифференцирование. Определение матричной производной. Градиентный спуск (GD), стохастический градиентный спуск (SGD), метод Ньютона. Примеры нахождения ММП оценок для многомерного нормального распределения.

1.1 Определение $Df(x)[\Delta x]$

$$Df(x)[\Delta x] = \lim_{t \to 0} \frac{f(x + t \cdot \Delta x) - f(x)}{t}$$

$Df(x)[\Delta x]$ обозначают дифференциал функции f (линейную часть приращения функции). В свою очередь производная функция f в точке x_0 – линейный оператор $Df(x)$, который лучше всего аппроксимирует приращение функции:

$$f(x + \Delta x) - f(x) = Df(x)[\Delta x] + \mathcal{O}(\Delta x)$$

В случае 1) $f : \mathbb{R}^{n \times 1} \to \mathbb{R}$**:**

$$Df(x)[\Delta x] = \langle \nabla f(x), \Delta x \rangle, \text{ где } \underbrace{\nabla f(x) = \left(\frac{\partial f}{\partial x_1}(x), \frac{\partial f}{\partial x_2}(x), \ldots, \frac{\partial f}{\partial x_n}(x) \right)^T}_{\text{градиент функции – столбец частных производных}} \in \mathbb{R}^{n \times 1}$$

Вектор приращения Δx тоже вектор столбец $\in \mathbb{R}^{n \times 1}$. Теперь пару примеров:

- $f(x) = a^T \cdot x: \ Df(x)[\Delta x] = a^T \cdot \Delta x = \langle a, \Delta x \rangle, \ \nabla f(x) = a$
- $f(x) = x^T A x: \ Df(x)[\Delta x] = x^T (A + A^T) \Delta x = \langle (A + A^T)x, \Delta x \rangle = \langle \nabla f(x), \Delta x \rangle$
- $f(x) = \frac{1}{2} ||Ax - b||_2^2: \ Df(x)[\Delta x] = \langle \nabla f(x), \Delta x \rangle, \ \nabla f(x) = A^T \cdot (Ax - b)$

В случае 2) $f : \mathbb{R}^{m \times n} \to \mathbb{R}$**:**

$$Df(x)[\Delta x] = \langle \nabla f(x), \Delta x \rangle, \text{ где } \underbrace{\nabla f(x) = \left(\frac{\partial f}{\partial x_{ij}}(x) \right)_{i=1,j=1}^{m,n}}_{\text{матрица частных производных}} \in \mathbb{R}^{m \times n}, \ \Delta x \in \mathbb{R}^{m \times n}$$

Скалярное произведение матриц: $\langle A, B \rangle = \text{tr}(A^T B)$

1.2 Основные формулы матричного дифференцирования

- $D(AX)[\Delta X] = \text{tr}(A^T \Delta X)$
- $D(X^{-1})[\Delta X] = -X^{-1} \Delta X X^{-1}$
- $DDetX[\Delta X] = DetX \cdot \text{tr}(X^{-1} \Delta X)$

1.3 Градиентный спуск (GD), стохастический градиентный спуск (SGD), метод Ньютона.

Направление максимального роста функции задаётся градиентом
Направление максимального падения функции задаётся антиградиентом

1. Градиентный спуск:
$$w^{(t+1)} = w^{(t)} - \eta \nabla L(w^{(t)})$$
$\eta > 0$ – шаг/темп обучения, t – итерация, L – функция потерь (или любая другая функция), w – по какому набору оптимизируем, $-\nabla L(w)$ – антиградиент функции потерь

2. Стохастический градиентный спуск отличается от простого градиентного тем, что мы ищем градиент не для $L(w) = \sum_{i=1}^{N} L_i(w)$, а для $L_R(w) = \sum_{i \in R} L_i(w)$, где R - случайное подмножество элементов выборки, называемое minibatch:
$$w^{(t+1)} = w^{(t)} - \eta \sum_{i \in R} \nabla L_i(w^{(t)})$$

3. Метод Ньютона:
$$w^{(t+1)} = w^{(t)} - H_{(t)}^{-1} \nabla L(w^{(t)})$$
$H(L)$ - матрица Гессе для функции потерь:
$$H(L) = \frac{\partial}{\partial w} \left(\frac{\partial L(w)}{\partial w} \right)^T = \begin{bmatrix} \frac{\partial^2 L}{\partial w_1^2} & \frac{\partial^2 L}{\partial w_1 \partial w_2} & \cdots & \frac{\partial^2 L}{\partial w_1 \partial w_d} \\ \frac{\partial^2 L}{\partial w_2 \partial w_1} & \frac{\partial^2 L}{\partial w_2^2} & \cdots & \frac{\partial^2 L}{\partial w_2 \partial w_d} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 L}{\partial w_d \partial w_1} & \frac{\partial^2 L}{\partial w_d \partial w_2} & \cdots & \frac{\partial^2 L}{\partial w_d^2} \end{bmatrix}$$
Откуда выводится формула: рассмотрим ряд Тейлора до второго порядка
$$L(w + \Delta w) = L(w) + g^T \cdot \Delta w + \frac{1}{2} \Delta w^T \cdot H \cdot \Delta w + \mathcal{O}(\|\Delta w\|^2)$$
$$\nabla_{\Delta w} L = g + H \cdot \Delta w = 0$$
$$\Delta w = -H^{-1} \cdot g$$

На примере множественной регрессии: модель $y = X\beta + \epsilon$, функция потерь:
$$L(\beta) = \sum_{i=1}^{N} (y_i - \langle x_i, \beta \rangle)^2 = (y - X\beta)^T (y - X\beta) = y^T y - 2y^T X\beta + \beta^T X^T X\beta$$
$$\nabla L(\beta) = -2X^T y + 2X^T X\beta$$
$$H = 2X^T X$$

1. GD: $\beta^{(t+1)} = \beta^{(t)} - \eta(-2X^T y + 2X^T X\beta^{(t)})$

2. SGD: случайно выбираем один элемент $i : (x_i, y_i)$ – minibatch из одного элемента,

$$L_i(\beta) = (y_i - \langle x_i, \beta \rangle)^2$$
$$\nabla L_i(\beta) = -2(y_i - \langle x_i, \beta \rangle)x_i$$
$$\beta^{(t+1)} = \beta^{(t)} - 2\eta(\langle x_i, \beta^{(t)} \rangle - y_i)x_i$$

3. Метод Ньютона: $\beta^{(t+1)} = \beta^{(t)} - \eta(2X^TX)^{-1}(-2X^T + 2X^TX\beta^{(t)})$

1.4 Пример нахождения ММП оценок для многомерного нормального распределения

Application: Let's consider the problem of finding the maximum likelyhood estimatiors for the multidimensional Normal distribution: $x_i \sim \mathcal{N}(\mu, \Sigma)$

$$p(x|\,\mu, \Sigma) = \frac{1}{\sqrt{Det(2\pi\Sigma)}} \exp(-\tfrac{1}{2}(x-\mu)^T\Sigma^{-1}(x-\mu))$$

The Likelyhood function will look tike that: $L(\mu, \Sigma) = \prod_{i=1}^{n} \frac{1}{\sqrt{Det(2\pi\Sigma)}} \exp(-\frac{1}{2}(x_i - \mu)^T\Sigma^{-1}(x_i - \mu)) = Det(2\pi\Sigma)^{-\frac{n}{2}} \exp(-\frac{1}{2}\sum_{i=1}^{n}(x_i - \mu)^T\Sigma^{-1}(x_i - \mu)) \to \max_{\mu,\Sigma}$

$$\log L = -\tfrac{n}{2}\log Det(2\pi) - \tfrac{n}{2}\log Det(\Sigma) - \tfrac{1}{2}\sum_{i=1}^{n}(x_i-\mu)^T\Sigma^{-1}(x_i-\mu) \to \max_{\mu,\Sigma}$$

$$\nabla_\mu \log L = -\tfrac{1}{2}\sum_{i=1}^{n} 2\Sigma^{-1}(x_i - \mu) = 0 \Rightarrow \boxed{\hat{\mu} = \frac{1}{n}\sum_{i=1}^{n} x_i}$$

$$\nabla_\Sigma \log L = \{\texttt{for convenience let } \Lambda = \Sigma^{-1}\} = \nabla_\Lambda \left(-\tfrac{n}{2}\log Det\Lambda^{-1} - \tfrac{1}{2}\sum_{i=1}^{n}(x_i-\mu)^T\Lambda(x_i-\mu)\right) = \tfrac{n}{2}\underbrace{\Lambda^{-T}}_{\Lambda^{-1}} - \tfrac{1}{2}\sum_{i=1}^{n}(x_i-\mu)(x_i-\mu)^T = 0$$

$$\Rightarrow \hat{\Lambda}^{-1} = \boxed{\hat{\Sigma} = \frac{1}{n}\sum_{i=1}^{n}(x_i-\mu)(x_i-\mu)^T}$$

2 Понятие явления переобучения. Способы борьбы с переобучением. Кросс-валидация, её виды.

Переобучение - ситуация, когда модель подстраивается под конкретную выборку, на которой она была обучена, и работает значительно хуже на любых других данных.

Как "увидеть" переобучение?

- Посмотреть на разделяющую поверхность или на линию регрессии **(1)**. На графиках переобученные модели, так как поверхности прямо по конкретным точкам:

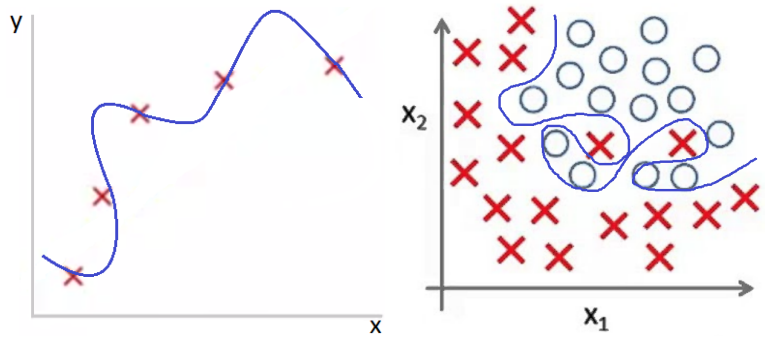


Рис. 1. Линия регрессии(слева) и разделяющая поверхность(справа)

- Посмотреть на разницу между метриками качества на обучении и контроле. Формально степень переобучения можно определить как разность $Metric_{train} - Metric_{test}$. Графически это можно представить на кривых обучения **(2)**:

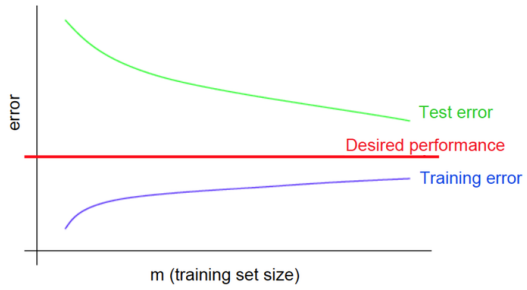


Рис. 2. Кривая обучения

Bias-Variance дилемма и переобучение:

Знаем, что можно разложить качество предсказания модели на составляющие (Bias-Variance decomposition). Ниже пример для метрики MSE:

$$E_{x^n}\left[E_{x,y}[(y-\mu(x^n)(x))^2]\right] = E_{x,y}[(y-E(y|x))^2]\ \textbf{(noise)}+$$
$$E_{x,y}[(E_(y|x) - E_{x^n}\mu(x^n)(x))^2]\ \textbf{(bias)}+$$
$$E_{x,y}\left[E_x^n\left[(\mu(x^n)(x) - E_{x^n}\mu(x^n)(x))^2\right]\right]\ \textbf{(variance)},$$

$\mu(x^n)(x)$ - это ответ модели, *обученной* на выборке из n элементов и *применённой* к наблюдению x. $E[]$ - матожидание выражения, $E[y|x]$ - матожидание y при условии x.

Из выражения видно, что bias - это отклонение нашей усреднённой модели от лучшей из моделей, а variance - это разброс между моделями по всем возможным выборкам и усреднённой модели.

Высокий variance отвечает за переобучение, т.к. отражает зависимость качества модели от выборки, по которой мы её построили.

Из bias-variance decomposition следуют некоторые методы борьбы с переобучением: bagging, boosting.

Как бороться с переобучением:

1. Увеличить количество данных, на которых мы подбираем параметры для модели:

   - Где-то найти ещё выборку
   - Bootstrap - *случайно с возвращением выбираем элементы из нашей выборки.* С помощью усреднённых метрик подбираем параметры.
   - Cross-validation - разбиваем выборку на блоки и используем один из них для контроля, а остальные для обучения. Проделываем это столько раз, сколько блоков, считаем метрику каждый раз и усредняем.

2. Снизить сложность модели:

   - Уменьшить количество фичей или (и) параметров
   - Использовать специфические для класса моделей методы. Пример: pruning - стрижка decission tree, когда мы сначала строим дерево, а потом удаляем узлы с листов. Тут подробнее.

3. Ограничить значения параметров в модели - добавить регуляризацию в наш Loss. Пример: LASSO (L1) и RIDGE (L2).

4. Комбинировать модели:

   - Bagging - агрегируем предсказания независимых моделей (должны быть либо разных видов, либо обучены на разных, независимых данных). Идея в том, что модели работают параллельно и из-за агрегирования корректируют друг друга. Пример: random forest - мы делаем много случайных подвыборок через bootstrap, применяем к ним деревья, у которых случайно выбираются фичи для построения. Их предсказания усредняем.
   - Boosting - последовательно обучаем несколько алгоритмов, чтобы каждый следующий исправлял ошибки предыдущего. Пример: градиентный бустинг.

Кросс-валидация - способ валидации модели, при котором мы делим нашу обучающую выборку на несколько частей и используем одну из частей для контроля, а остальные для обучения модели. Пройдясь по всем частям в цикле и вычислив метрику, получим среднюю точность модели, посчитанную на большем количестве данных, чем первоначальное.

Источник всей инфы далее (много графиков для наглядности и примеры кода)

Принципы разбиения выборки (для любого способа валидации):

1. валидация моделирует реальную работу алгоритма. Следствие: при разбиении выборки должны сохраняться пропорции классов. Пример применения: если алгоритм должен работать на новых пользователях, по которым пока нет данных, то в валидационной выборке должны быть такие пользователи.

2. нельзя явно или неявно использовать метки объектов, на которых оцениваем ошибку (качество). Пример применения: мы сначала разбиваем на обучение и валидацию, и только потом делаем target-encoding, потому что иначе в нашей

выборке для обучения неявно будет использоваться информация о таргетах контроля.

3. тест должен быть случайным (или специально подготовленным)

Принципы кросс-валидации: 1) сохраняем баланс классов, 2) рандомизируем, когда разбиваем на части.

Виды кросс-валидации:

- K-fold - алгоритм:
  1. Делим выборку на k примерно равных частей
  2. Цикл по i=1,...k: используем i-ю часть для вычисления ошибки, объединение остальных для обучения.
  3. усредняем полученные k ошибок
- Leave-one-out - K-fold, где k=количеству наблюдений в выборке
- LeaveOneGroupOut - алгоритм:
  1. Делим выборку на несколько групп в данных (по значениям фичей или по каким-то содержательным соображениям)
  2. Цикл по i=1,...количество выделенных групп: используем i-ю группу для вычисления ошибки, а остальные для обучения модели.
  3. Усредняем полученные ошибки (можно усреднять с весами, пропорциональными размеру группы в выборке, если есть необходимость)
- Out-of-time-контроль - кросс-валидация для временных рядов **(3)**

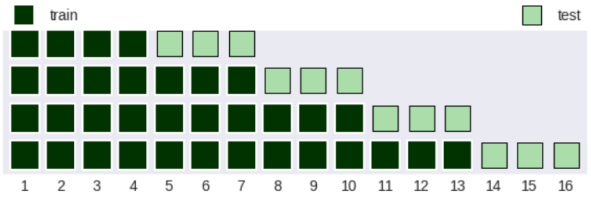


Рис. 3. Кросс-валидация временных рядов

Нюансы K-fold: с увеличением k

- надёжнее оценка качества модели
- обучающая выборка больше походит на остальные данные
- время контроля линейно возрастает
- надо следить за метрикой качества, т.к. не любая адекватно оценивает качество на маленьких подвыборках (хз, какой пример привести)

3 Метрические методы классификации и регрессии. Проклятье размерности.

3.1 Алгоритм k Nearest Neighbours

.

Постановка 1. *Пусть есть выборка* $\{x_i\} \in \mathbb{X}$*, и множество непересекающихся классов* $\Upsilon = \{1, 2, \ldots, l\}$*, а также метрика* ρ*, и необходимо классифицировать новый объект* $u \in \mathbb{X}$

1. Вычислить и расположить элементы в порядке возрастания $\rho(u, x_u^{(1)}) \leq \rho(y, x_u^{(2)}) \leq \cdots \leq \rho(u, x_u^{(\ell)})$
2. Объект относится к тому классу, представителей которого окажется больше всего среди k его ближайших соседей

$$a(u; \mathbb{X}^{(\ell)}, k) = \arg_{y \in \Upsilon} \max \sum_{i=1}^{k} \cdot [y_u^{(i)} = y] \tag{1}$$

Гиперпараметры:

- $k \in \mathbb{N}$.
- Выбор метрики для расстояния между объектами.
- Если используется ядро, ему тоже надо придумать форму.
- При взвешенной регрессии в уравнение (1) добавляется вес w_i, об этом дальше здесь.

В невзвешенной регрессии предсказания вычисляются согласно формуле $pred(y_i|x_i \in \mathbb{N}(x)) = \text{argmax}_{a_i}(\sum_{t=1}^{k} \mathbb{I}(y(x_t) = a))$.

3.1.1 Преимущества

- Простой для ручной реализации.

3.1.2 Недостатки

- Алгоритм действует полным перебором вариантов и не содержит обучения (т.е. изменения самого себя в зависимости от выборки), lazy-learner.
- Куча гиперпараметров, для которых даже сетку не настроить, чтобы их одновременно гипероптимизировать.
- Вычислительная сложность классификации - $O(N \cdot n)$, где N - размер выборки и n - размерность пространства $\mathbb{X}$. Статья Вебера [1] показывает, что быстрее чем за линейное время ближайшего соседа просто не удастся отыскать.

3.1.3 k Centroids

Для каждого класса вычисляются только "центры" $c_{ij} = \frac{1}{|\{i:y_i=j\}|} \cdot \sum_{i,y_i=j} x_i$ и только они хранятся.

- Не требует экспоненциального количества времени и памяти для хранения, т.к. При этом информация о принадлежности ближайшего окружения точки игнорируется.
- С другой стороны, могут быть классы, чьё множество в исходном пространстве имеет формы типа кольца, и центроиды не несут никакой информации о таких классах.

3.2 Регрессия kNN

.

Так как таргет в kNN-регрессии можно упорядочить, то при построении с помощью алгоритма kNN можно говорить о предпочтении ± "плавной" зависимости, для которой естественен приоритетный учёт ближайших наблюдений, для kNN-регрессии характерны **весовые схемы** (weighted kNN Generalisations).

Во взвешенной регрессии функционал окажется таким:

$$Q(\alpha, X^l) = \sum_{i=1}^{l} w_{i(x)} \cdot (a - y_i)^2 \to \min_{\alpha \in \mathbb{R}}$$

Предсказания вычисляются по формуле Надарая-Ватсона:

$$\hat{y_x} = \frac{\sum_{t=1}^{k} w_t \cdot y(x_t)}{\sum_{t=1}^{k} w_t} \tag{2}$$

Популярными являются весовые схемы:

- $w_t = (k - t + 1)^\delta, \delta > 0$
- $w_t = \frac{1}{t^\delta}$
- Ядра: $w_t = K(\frac{\rho(x,x_t)}{h(x)})$, где $h(x)$ - **возрастающая** по x функция, которая придаёт различие любым различиям в малых расстояниях. Функция K должна давать самые большие значения для самых маленьких аргументов ρ (т.е. ситуации, когда точки совпадают) и околонулевые - для самых маленьких. Например, K может быть просто функцией нормального стандартного распределения:

$$K(x) = \frac{1}{\sqrt{2\pi}} \cdot \exp\left(\frac{-x^2}{2}\right) \tag{3}$$

.

3.3 Проклятие размерности †

Неформальное определение 1. ***Проклятием размерности*** † *(обычно) называется феномен экспоненциального роста необходимого количества памяти и времени на вычисления при "наивных" алгоритмах классификации, где количество необходимых сравнений расстояний между точками растёт полиномиально/экспоненциально (а не линейно, как при логистической регрессии), что сильно удлиняет их работу.*

4 Решающие деревья. Критерии расщепления, выбор оптимальных порогов. Способы борьбы с переобучением в решающих деревьях. Обработка категориальных признаков, обработка пропусков.

Рассмотрим бинарное дерево, в котором:

- каждой внутренней вершине v приписана функция (предикат) $\beta_v : \mathbb{X} \to \{0, 1\}$, обычно $\beta_v(x; j,t) = [x_j < t]$ – бинарное правило;
- каждой листовой вершине v приписан прогноз $c_v \in Y$ (в случае с классификацией листу также может быть приписан вектор вероятностей).

Рассмотрим теперь алгоритм $a(x)$, который стартует из корневой вершины v_0 и вычисляет значение функции β_{v_0}. Если оно равно нулю, то алгоритм переходит в левую дочернюю вершину, иначе в правую, вычисляет значение предиката в новой вершине и делает переход или влево, или вправо. Процесс продолжается, пока не будет достигнута листовая вершина; алгоритм возвращает тот класс, который приписан этой вершине. Такой алгоритм называется **бинарным решающим деревом**.

Критерий расщепления:

$$Q(R, \theta) = H(R) - \frac{|R_{left}|}{|R|} \cdot H(R_{left}) - \frac{|R_{right}|}{|R|} \cdot H(R_{right}), \text{ где меры неоднородности}$$

- для классификации:
  1. $H(R) = \sum_{k=1}^{K} p_k \cdot (1 - p_k)$ – критерий Джини
  2. $H(R) = -\sum_{k=1}^{K} p_k \cdot \log p_k$ – энтропийный критерий
  3. $H(R) = 1 - \max\limits_{1,\dots,K} p_k$ – miss-classification criteria
- для регрессии:
  1. $H(R) = \frac{1}{|R|} \sum_{(x_i,y_i)\in R} \left(y_i - \frac{1}{|R|} \sum\limits_{(x_j,y_j)\in R} y_j \right)^2$ – MSE
  2. $H(R) = \frac{1}{|R|} \sum_{(x_i,y_i)\in R} \left| y_i - \underset{(x_j,y_j)\in R}{Median}\ y_j \right|$ – MAE

Напоминание: параметр θ обозначает рассматриваемое правило для ветвления.

4.1 Методы борьбы с переобучением:

1. Критерии останова

   Эти критерии позволяют не достраивать дерево до самого конца, когда в каждом листе останется ровно по одному элементу.

   - Ограничение максимальной глубины дерева
   - Ограничение минимального числа объектов в листе
   - Ограничение максимального количества листьев в дереве
   - Останов в случае, если все объекты в листе относятся к одному классу
   - Требование, что функционал качества при дроблении улучшался как минимум на s процентов

2. Стрижка дерева

   Стрижка дерева является альтернативой критериям останова, описанным выше. При использовании стрижки сначала строится переобученное дерево (например, до тех пор, пока в каждом листе не окажется по одному объекту), а затем производится оптимизация его структуры с целью улучшения обобщающей способности.

   Одним из методов стрижки является cost-complexity pruning. Обозначим дерево, полученное в результате работы жадного алгоритма, через T_0. Поскольку в каждом из листьев находятся объекты только одного класса, значение функционала $R(T)$ будет минимально на самом дереве T_0 (среди всех поддеревьев). Однако данный функционал характеризует лишь качество дерева на обучающей выборке, и чрезмерная подгонка под нее может привести к переобучению. Чтобы преодолеть эту проблему, введем новый функционал $R_\alpha(T)$, представляющий собой сумму исходного функционала $R(T)$ и штрафа за размер дерева:

   $R_\alpha(T) = R(T) + \alpha|T|,$

   где $|T|$ — число листьев в поддереве T, а $\alpha > 0$ — параметр. Это один из примеров регуляризованных критериев качества, которые ищут баланс между качеством классификации обучающей выборки и сложностью построенной модели. Можно показать, что существует последовательность вложенных деревьев с одинаковыми корнями:

   $T_K \subset T_{K-1} \subset ... \subset T_0,$

   здесь T_K — тривиальное дерево, состоящее из корня дерева T_0, в которой каждое дерево T_i минимизирует критерий $R_\alpha(T)$ для α из интервала $\alpha \in [\alpha_i, \alpha_i + 1)$, причем

   $0 = \alpha_0 < \alpha_1 < ... < \alpha_K < \infty.$

   Эту последовательность можно достаточно эффективно найти путем обхода дерева. Далее из нее выбирается оптимальное дерево по отложенной выборке или с помощью кросс-валидации.

3. Использование в ансамблях (например, случайный лес)

4.2 Обработка категориальных признаков

Категориальный признак $x_j \in categories \equiv \{cat_1, \ldots, cat_k\}$. Его значение для конкретного наблюдения из выборки (x_j^i) означает определенную категорию, к которой наблюдение причислено. Мы хотим заменить каждое значение категориального признака некоторым числом, иными словами – закодировать, чтобы далее строить дерево с использованием привычных сплитов по числовым фичам.

- **Target Encoding**

  Суть: Значения категориальной переменной кодируется средним таргетом по категориям $encoding(cat_j) = \frac{\sum\limits_i [x_j^i = cat_m] \cdot y_i}{\sum\limits_i [x_j^i = cat_m]}$. Фактически, $encoding : category \in categories \mapsto value \in \mathbb{R}$.
  Проблема: При такой кодировке возникает target leakage (информация о таргете передается в фичи). Мы строим дерево на фичах, которые содержат в себе информацию о таргете, что ведет к переобучению.
  Решение проблемы: Подбираем числовое значение для кодировки определенной категории с помощью кросс-валидации. Разбивем на фолды и для каждого фолда считаем число для кодировки значения категории по описанному выше принципу, но уже не по всему датасету, а по остальным фолдам. И так делается для каждого фолда. Можно делать иначе и просто добавлять шум, но этот способ может быть хуже.

- **Label Encoding**

  Суть: Значения категориальной переменной кодируется порядковым номером. В лекции было сказано, что принцип особо не важен, например: в таком порядке в каком категории встречаются в датасете, такой номер и присваивается конкретной категории.
  Проблема: При такой кодировке возникает порядок там, где мы его не ожидаем, что может влиять на предсказательную силу обучаемого дерева.
  Комментарий: Кажется, сюда подойдет то, что написано в файле Соколова про обработку категориальных переменных. Можно упорядочить категории в фиче x_j по среднему значению таргета внутри категории, то есть $cat_{(1)}, \ldots, cat_{(k)}$: $\frac{\sum\limits_i [x_j^i = cat_{(1)}] \cdot y_i}{\sum\limits_i [x_j^i = cat_{(1)}]} \leq \ldots \leq \frac{\sum\limits_i [x_j^i = cat_{(k)}] \cdot y_i}{\sum\limits_i [x_j^i = cat_{(k)}]}$. Дальше кодируем эти категории по принципу $cat_{(i)} \mapsto i$ и можем проводить разбиения вершин как с обычными числовыми признаками. В файле Соколова указано, что так можно делать и внутри конкретной вершины дерева.

- **One Hot Encoding**

  Суть: Фактичеки, это кодирование дамми переменными. То есть если для признака x_j есть всего k значений категорий, то есть $x_j \in categories \equiv \{cat_1, \ldots, cat_k\}$, то создается k новых признаков, каждый из которых является индикатором $(x_{ohe_m} \equiv [x_j = cat_m], m \in \{1, \ldots, k\})$.
  Проблема: Данные могут сильно разростаться. Кроме этого, в каждой новой фиче, которая по сути вляется индикатором определенной категории, обычно достаточно мало единиц и больше нолей, что влияет на качество построения

дерева (а конкретно в момент подсчета функционала качества при разбиении вершин).

4.3 Обработка пропущенных значений

- *Игнорируем пропуски*

  Пусть нам нужно вычислить функционал качества для предиката $\beta(x) = [x_j < t]$, но в выборке R для некоторых объектов не известно значение признака j — обозначим их через V_j.

  В таком случае при вычислении функционала можно просто проигнорировать эти объекты, сделав поправку на потерю информации от этого:

  $Q(R, j, s) \approx \frac{|R\ V_j|}{|R|} Q(R\ V_j, j, s)$.

  Затем, если данный предикат окажется лучшим, поместим объекты из V_j как в левое, так и в правое поддерево. Также можно присвоить им при этом веса $\frac{|R_l|}{|R|}$ в левом поддереве и $\frac{|R_r|}{|R|}$ в правом. В дальнейшем веса можно учитывать, добавляя их как коэффициенты перед индикаторами $[y_i = k]$ во всех формулах.

- *Обработка пропусков при прогнозировании*

  Допустим, нам надо спрогнозировать таргет для наблюдения, у которого значение одной из фичей пропущено. Если объект попал в вершину, предикат которой не может быть вычислен из-за пропуска, то прогнозы для него вычисляются в обоих поддеревьях, и затем усредняются с весами, пропорциональными числу обучающих объектов в этих поддеревьях. Иными словами, если прогноз вероятности для класса k в поддереве R_m обозначается через $a_{mk}(x)$, то получаем такую формулу:

  $$a_{mk}(x) = \begin{cases} a_{\ell k}(x), & \beta_m(x) = 0 \\ a_{rk}(x), & \beta_m(x) = 1 \\ \frac{|R_\ell|}{|R_m|} a_{\ell k}(x) + \frac{|R_r|}{|R_m|} a_{rk}(x), & \beta_m(x) \text{ нельзя вычислить.} \end{cases}$$

  Либо можно просеить наблюдение по другому признаку. То есть у нас есть обученное дерево, просеивая через него объект мы сталкиваемся с проблемой в одной из вершин, что дальше мы просеить его не можем, так как значение признака, который используется в предикате в этой вершине, пропущено для нашего наблюдения. Тогда для этой вершины можно построить предикат с использованием другого признака (значение которого не пропущено для наблюдения, таргет которого надо предсказать), который даст похожее разбиение наблюдений из обучающей выборки в этой вершине. Такой предикат можно использовать и он называется "суррогатный предикат".

- *Простые методы*

  Иногда гораздо более простые методы показывают не менее качественные результаты: замена пропущенных значений на ноль; на число, которое больше максимального по фиче.

5 Линейные методы классификации и регрессии. Определение отступа. Различные виды верхних оценок на функционал доля ошибок. Логлосс, Hinge-loss. Обобщения линейных моделей для задачи классификации на K классов. Подходы One-vs-one и one-vs-all

5.1 Линейная регрессия

Алгоритм выглядит следующим образом:

$$a(x) = \langle w,x_i \rangle + b$$

где $w,x \in \mathbb{R}^d$, $b = const$. Если предполагается наличие константного признака в X, то можно переписать $a(x) = \langle w,x_i \rangle$ (или $a(x) = w^T x$). Алгоритм настраивается путем минимизации суммы квадратов отклонений (Родной метод наименьших квадратов):

$$RSS = \sum_{i=1}^{n} e_i^2 = \sum_{i=1}^{n} (y_i - a(x_i))^2 = \sum_{i=1}^{n} (y_i - \langle w,x_i \rangle)^2 \to \min_w$$

В результате оптимизации получаем веса: $w = (X^T X)^{-1} X^T y$ Частный случай для регрессии с одним x и константой: $w = \dfrac{cov(x,y)}{var(x)}$ При множественной регрессии может возникать проблема вырожденности матрицы $w = (X^T X)^{-1} X^T y$, которая может быть решена следующими способами:

- Регуляризация - например, с l2-регуляризацией $w = (X^T X + \lambda I)^{-1} X^T y$
- Отбор признаков - умный перебор подмножества признаков, оценка качества признаков (фильтры), встроенные методы (ex: LASSO)
- Уменьшение размерности (например, PCA)
- Увеличение выборки

5.2 Линейная классификация

Базовый алгоритм линейной классификации: $a(x) = sign(\langle w,x_i \rangle + b)$, где $w,x \in \mathbb{R}^d$, $b = const$. При наличии константного признака все аналогично регрессии. Это гиперплоскость, которая разделяет признаковое пространство на 2 части. Почему гиперплоскость: рассмотрим $w_1 x_i^{(1)} + w_1 x_i^{(2)} + ... + w_n x_i^{(n)} + b = 0$. Здесь можно выразить один из иксов через остальные (переносим вправа и делим на w). Соответственно остается n-1 независимых иксов, поэтому размерность разделяющей поверхности меньше на 1, чем всего пространства, а значит это гиперплоскость. Вполне логично настраивать такую модель классификации по доле ошибок (рассмотриваем вариант, где

$y = -1,1$):

$$Q = \frac{1}{n}\sum_{i=1}^{n}[a(x_i) \neq y_i] = \frac{1}{n}\sum_{i=1}^{n}[sign(\langle w,x_i\rangle) \neq y_i] = \frac{1}{n}\sum_{i=1}^{n}[y_i * sign(\langle w,x_i\rangle) \neq y_i^2] =$$
$$= \frac{1}{n}\sum_{i=1}^{n}[y_i * sign(\langle w,x_i\rangle) \neq 1] = \frac{1}{n}\sum_{i=1}^{n}[y_i * (\langle w,x_i\rangle) < 0] \quad (4)$$

Последний переход возможен, так как раз $y_i * sign(\langle w,x_i\rangle) \neq 1$, значит он равен -1, а значит меньше 0, а функцию sign можно убрать, так как она меньше 0 когда ее аргумент меньше 0.

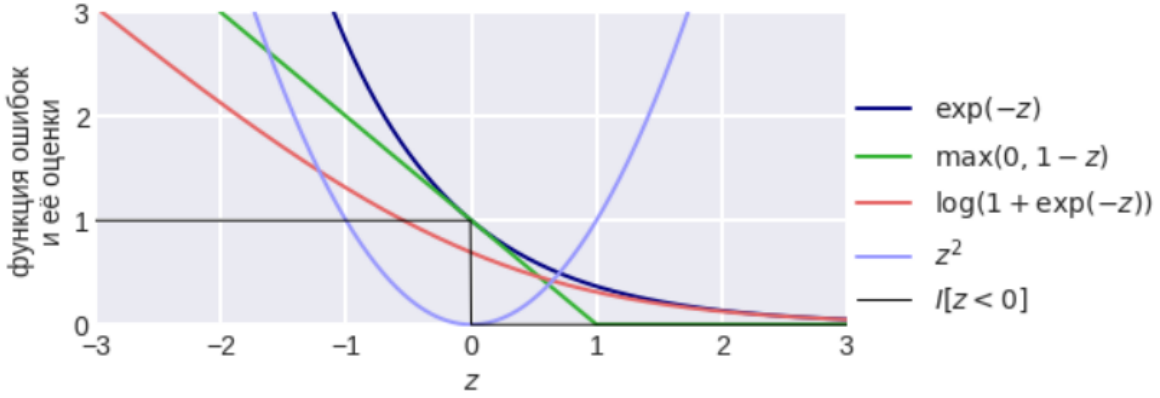
5.3 Определение отступа

(Из Соколова) Величина $M_i = y_i\langle w, x_i\rangle$, называемая отступом (margin). Знак отступа говорит о корректности ответа классификатора (положительный отступ соответствует правильному ответу, отрицательный — неправильному), а его абсолютная величина характеризует степень уверенности классификатора в своём ответе. Напомним, что скалярное произведение $\langle w, x_i\rangle$пропорционально расстоянию от разделяющей гиперплоскости до объекта; соответственно, чем ближе отступ к нулю, тем ближе объект к границе классов, тем ниже уверенность в его принадлежности.

5.4 Различные виды верхних оценок на функционал доля ошибок. Логлосс, Hinge-loss

Полученный выше функционал $[M_i < 0]$ невозможно оптимизировать с помощью, например, градиентного спуска, так как он дискретный. Вместо него работают с верхними оценками - гладкими функциями, среди которых выделяют:

- Log-loss - $log(1 + e^{-M})$ (выводится из логистической регрессии, см. 6 билет)
- Hinge-loss - $max(0, 1 - M)$ (выводится из Soft-margin постановки SVM, см. 8 билет)



5.5 Обобщения линейных моделей для задачи классификации на K классов.

Одним из вариантов многоклассовой классификации является ее сведение к бинарной. Для этого используют подходы one-versus-all и all-versus-all.

- One-versus-All: обучаем K классификаторов (K - количество классов), в каждом из которых выдаем вероятность принадлежности к данному классу (против остальных). На этапе предсказания для нового объекта присваиваем ему тот класс, у которого получилась наибольшая вероятность (чтобы вероятности в сумме давали 1, нужен softmax, но это не влияет на предсказание). Основная проблема этого подхода в том, что возникает дисбаланс классов. Из-за этого для правильности модели лучше подбирать веса, более чувствительные к даннному k-му классу, плюс не все метрики качества можно использовать (например, вместо ROC-AUC лучше PR-AUC).

- All-versus-All (это версия Соколова, в наших лекциях называется One-versus-One): обучаем C_K^2 классификаторов, которые сравнивают классы попарно. Как делать предикт: выбираем тот класс, который чаще всего побеждает при попарных сравнениях. Плюс этого подхода по сравнению с предыдущим в том, что здесь мощности классов сопоставимы.

6 Логистическая регрессия. Вывод формулы логистической регрессии. Регуляризация функционала в логистической регрессии. Заставляет ли функционал в задаче логистической регрессии предсказывать вероятности?

6.1 Определение логистической регрессии

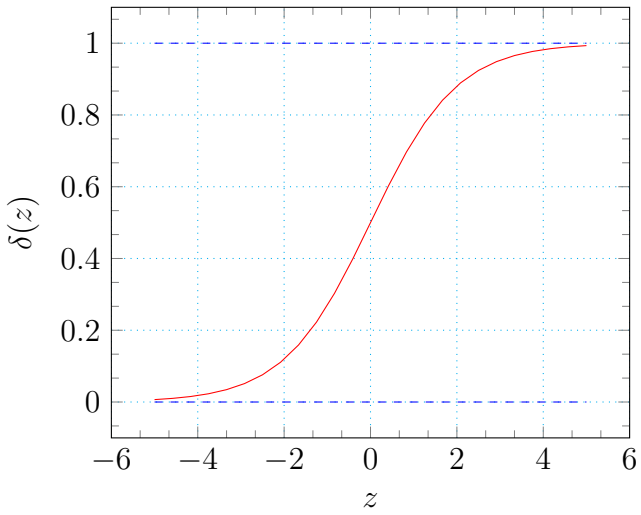
Логистическая регрессия - это метод обучения, который получается в результате использования логистической функции потерь.

Мы решаем задачу принадлежности объекта к классу $+1$: $a(x) = p(y = +1|x)$

Зададим классификатор следующим образом: $a(x) = \delta(\langle w, x\rangle) = \frac{1}{1+\exp(-\langle w,x\rangle)}$, где соответственно функция сигмоиды выглядит следующим образом: $\delta(z) = \frac{1}{1+\exp(-z)}$. Линии уровня сигмоиды - гиперплоскости, поэтому данный классификатор является линейным.

График сигмоиды выглядит следующим образом:

График сигмоиды



Ответы классификатора $a(x) \in (0,1)$ - их можно интерпретировать как вероятность принадлежности к классу $+1$.

6.2 Вывод формулы логистической регрессии: логарифм правдоподобия

Правдоподобие с точки зрения классификатора:

$$L = \prod_{i=1}^{n} a(x_i)^{[y_i=1]} * (1 - a(x_i))^{[y_i=-1]}$$

Прологарифмируем:

$$\log L = \sum_{i=1}^{n}([y_i = 1] * \log a) + [y_i = -1] * \log(1 - a(x_i))) =$$

На следующем шаге вместо $a(x_i)$ вставим сигмоиду и вынесем минус из-под логарифма в первом слагаемом, во втором слагаемом под логарифмом все приведем под общий знаменатель:

$$= \sum_{i=1}^{n}(-[y_i = 1] * \log(1 + \exp(-\langle w, x_i\rangle) + [y_i = -1] * \log(\frac{\exp(-\langle w,x_i\rangle)}{1 + \exp -\langle w,x_i\rangle})) =$$

Во втором слагаемом под логарифмом домножим числитель и знаменатель на $\exp \langle w,x_i\rangle$ и выносим степень -1 тоже. В итоге минус за сумму вынесется.

$$= -\sum_{i=1}^{n}([y_i = 1] * \log(1 + \exp(-\langle w, x_i\rangle)) + [y_i = -1] * \log(1 + \exp(\langle w, x_i\rangle))) =$$

$$= -\sum_{i=1}^{n} \log(1 + \exp(-y_i\langle w, x_i\rangle)) \to \max_w$$

Уберем минус и получим Loss-функцию:

$$\tilde{Q} = \sum_{i=1}^{n} \log(1 + \exp(-\underbrace{y_i\langle w, x_i\rangle}_{Margin_i})) \to \min_w$$

6.3 Решение задачи минимизации Loss-функции

Аналитическое решение $\nabla_w \tilde{Q} = 0$ не найдется, поэтому будем искать численное решение:

- Метод градиентного спуска:

$$w_{k+1} = w_k - \alpha \nabla_w \tilde{Q}(w_k)$$

Вспомним, что: $\nabla_w \langle w,x_i \rangle = x_i$, $\nabla_a (a^T x a) = (x + x^T)a$

$$\nabla_w \tilde{Q} = \frac{1}{n} \sum_{i=1}^{n} \frac{-\exp(-y_i \langle w,x \rangle) y_i x_i}{1 + \exp(-y_i \langle w,x \rangle)} = \frac{1}{n} \sum_{i=1}^{n} -y_i x_i \frac{1}{1 + \exp(y_i \langle w,x \rangle)} = \frac{1}{n} \sum_{i=1}^{n} -y_i x_i \delta(-y_i \langle w,x \rangle)$$

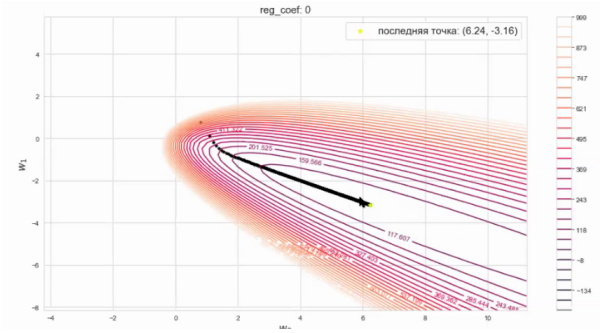
- Метод Ньютона:

$$w_{k+1} = w_k - \alpha H^{-1} \nabla_w \tilde{Q}(w_k)$$

$$H = \frac{\delta^2 \tilde{Q}(w)}{\delta w_k^{(i)} w_k^{(j)}}$$

6.4 Регуляризация функционала в логистической регрессии

При решении, например, методом градиентного спуска, можно столкнуться с проблемой наличия плато. Плато - область, в которой функция примерно постоянна и везде достигает своего минимума. Его стоит изберать - иначе во-первых, метод градиентного спуска будет долго сходиться и, во-вторых, так значения функционала примерно одинаковые и шаги градиентного спуска слишком маленькие, метод никуда не выйдет. Плато выглядят так:



Как бороться с плато? Используем регуляризацию:

$$\tilde{Q} = \sum_{i=1}^{n} \log(1 + \exp(-y_i \langle w, x_i \rangle)) + \underbrace{\frac{\lambda}{2} \parallel w \parallel_2^2}_{l_2\text{-регуляризация}} \to \min_w$$

Для чего нужна регуляризация?

- Регуляризация борется с плато - делает линии уровня функционала более кругообразными

- Допустим, найдена гиперплоскость, которая идеально разделяет множества. Будем умножать ее веса w на увеличивающиеся положительные величины $w \to \infty$ - гиперплоскость не изменится. Тогда $-y_i\langle w, x\rangle \to \infty$. При этом функционал (без регуляризации) будет уменьшаться $\tilde{Q} = \sum_{i=1}^{n} \log(1 + \exp(-y_i\langle w, x_i\rangle)) \to 0$. С точки зрения нашей задачи ничего не изменится (гиперплоскость). Если добавить регуляризацию, то при $w \to \infty$, тогда $\frac{\lambda}{2} \parallel w \parallel^2 \to \infty$ - поэтому у нас не будут бесконечно увеличиваться веса

Кроме того, есть l_1-регуляризация.

$$\tilde{Q} = \sum_{i=1}^{n} \log(1 + \exp(-y_i\langle w, x_i\rangle)) + \underbrace{\frac{\lambda}{2} \parallel w \parallel_1}_{l_1\text{-регуляризация}} \to \min_w$$

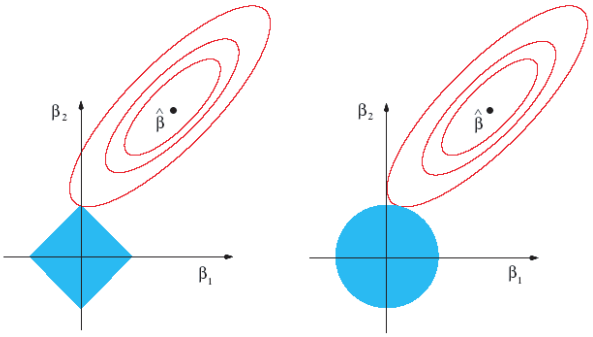
- Для l_1-регуляризации мы используем Манхеттенскую норму: $\parallel w \parallel_1 = \sum_{j=1}^{d} |w_j|$.

  В таком случае точка решения будет лежать в одном из углов ромба (см. картинку ниже - слева). Она позволяет отбирать признаки - какая-та часть коэффициентов будет равна 0 (это случится если не слишком маленькая лямбда, в противном таком случае - ромб будет большой, залезет на овал и там будет оптимум, то есть признаки не занулятся)

- Для l_2-регуляризации мы используем Евклидово расстояние. Точка решения будет лежать в любом месте на окружности (см. картинку ниже - справа).

- Еще один вид регуляризации: elsatic net (нечасто используется). Линии уровня выглядят как юла:

$$\tilde{Q} = \sum_{i=1}^{n} \log(1 + \exp(-y_i\langle w, x_i\rangle)) + \underbrace{\gamma_1 \parallel w \parallel_1 + \gamma_2 \parallel w \parallel_2^2}_{elasticnet} \to \min_w$$



Для l_2-регуляризации найдем градиент от регуляризатора $\tilde{Q}$:

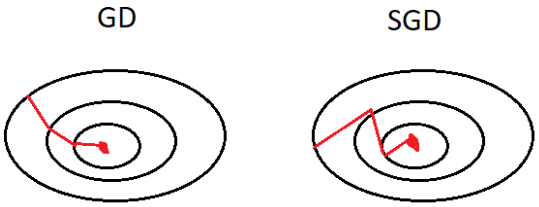
$$\nabla_w \tilde{Q} = \frac{1}{n} \sum_{i=1}^{n} -y_i x_i \delta(-y_i\langle w,x\rangle) + \lambda w$$

Аналогично для стохастического градиентного спуска:

$$\tilde{\nabla}_w \tilde{Q} = \frac{1}{|N|} \sum_{i \in N} -y_i x_i \delta(-y_i \langle w,x \rangle) + \lambda w$$

где N - элементы батча. SGD может блуждать больше, чем GD (см. картинку ниже), но в его направление аналогичное. Скорость схождения у SGD примерно такая же, как и у GD, иногда слегка больше. Если объектов слишком много, то GD нереализуем (не хватит оперативной памяти), т таких случаях помогает SGD.

Различия в поисках решений наглядно выглядят так:



6.5 Заставляет ли функционал в задаче логистической регрессии предсказывать вероятности?

Да, заставляет

Функция потерь Log-loss выглядит так:

$$-\sum_{i=1}^{l} ([y_i = -1] \log a(x_i) + [y_i = +1] \log(1 - a(x_i))) \to \min$$

Запишем математическое ожидание log-loss функции в точке x:

$$\mathbb{E}\Big[L(y,a)|x\Big] = \mathbb{E}\Big[-([y_i = -1] \log a - [y_i = +1] \log(1-a))|x\Big] =$$

$$= -p(y = +1|x) \log a - (1 - p(y = +1|x)) \log(1 - a)$$

Продифференцируем данное выражение по a:

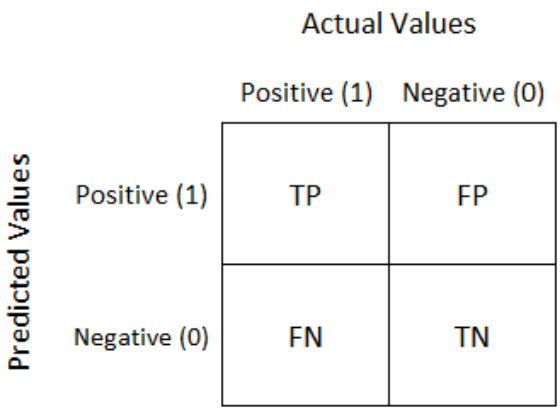
$$\frac{\delta}{a}\Big[L(y,a)|x\Big] = -\frac{p(y = +1|x)}{a} - \frac{1 - p(y = +1|x)}{1 - a} = 0$$

Получается, что $a* = p(y = +1|x)$, из чего можно сделать вывод о том, что оптимальный ответ алгоритма равен вероятности принадлежности к классу $+1$

7 Метрики качества классификации и регрессии. Матрица ошибок. Метрики в случае многоклассовой задачи классификации. Рок-кривая, площадь под ней, явная формула для рок-кривой.

7.1 Матрица ошибок

Матрица ошибок – матрица, размера KxK, где K – количество классов, которая показывает точность классификации модели. В случае двух классов:



Многоклассовый случай. Пусть есть выборка X размера n, y_i – метки класса для каждого объекта i (всего K-штук классов). Матрицей ошибок называется следующая матрица:

$$M = m_{jk}, \text{ размерности KxK}$$

$$m_{jk} = \sum_{i=0}^{N} [a(x_i) = j][y_i = k], \text{ где j – класс, предсказанный классификатором.}$$

7.2 Метрики качества для задач регрессии

- MSE $= \frac{1}{n} \sum_{i=1}^{n} (y_i - a_i)^2$
- RMSE $= \sqrt{MSE}$
- MAE $= \frac{1}{n} \sum_{i=1}^{n} w_i \cdot |y_i - a_i|$
- $-\log(L) = -\log(\text{Функция правдоподобия})$
- MAPE $= \frac{1}{n} \sum_{i=1}^{n} \frac{|y_i - a_i|}{y_i}$

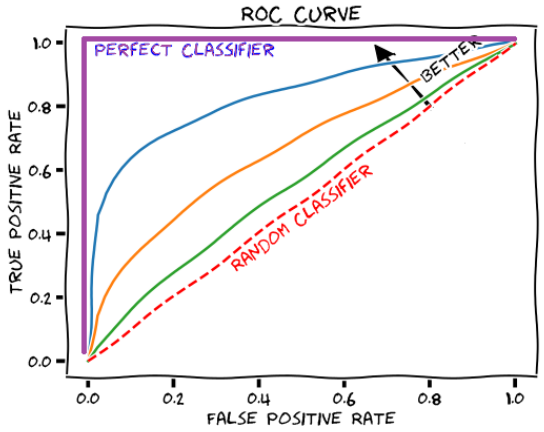
7.3 Метрики качества для задач классификации

- Функция правдоподобия $= \prod_{i=1}^{n} a_i^{y_i} * (1 - a_i)^{1-y_i}$

- Accuracy $= \frac{TP+TN}{TP+TN+FP+FN}$
- Precision $= \frac{TP}{TP+FP}$
- Recall $= \frac{TP}{TP+FN}$
- f1-measure $= \frac{2}{\frac{1}{precision}+\frac{1}{recall}}$
- ROC-AUC (ниже)

7.4 ROC кривая

ROC кривая – кривая зависимости *False Positive Rate* (FPR) $= \frac{FP}{FP+TN}$ и *True Positive Rate* (TPR) $= \frac{TP}{TP+FN}$. AUC ROC – площадь под этой кривой, значение характеризует долю правильно классифицируемых пар. Для ее построения ранжируем вероятность присвоения класса ($b(x)$), для каждого порога считаем FPR и TPR.



$$AUC = \frac{\sum_i \sum_j [y_i < y_j] \times [a(x_i) < a(x_j)]}{\sum_i \sum_j [y_i < y_j]} = \frac{1}{n_- \times n_+} \times \sum_{i<j} [y_i < y_j]$$

ROC может пойти по диагонали (не ступенькой), если у одинаковых наблюдений будут разные метки. Самый плохой случай AUC: AUC $= 0.5$, т.к. по сути ничего не упорядочили и угадываем в 50% случаев. Если AUC ROC $= 0$, то значит, что присвоенные метки полностью перепутаны и достаточно поменять их местами.

Критерий AUC-ROC имеет большое число интерпретаций: например, он равен вероятности того, что случайно выбранный положительный объект окажется позже случайно выбранного отрицательного объекта в ранжированном списке, порожденном $b(x)$.

Кривая ROC не всегда ступенчатая: точки могут соединяться и наклонными линиями.

7.5 Метрики в случае многоклассовой задачи классификации

В многоклассовых задачах, как правило, стараются свести подсчет качества к вычислению одной из рассмотренных выше двухклассовых метрик. Выделяют два подхода к такому сведению: микро- и макро-усреднение.

Пусть выборка состоит из K классов. Рассмотрим K двухклассовых задач, каждая из которых заключается в отделении своего класса от остальных, то есть целевые значения для k-й задачи вычисляются как $y_i^k = [y_i = k]$. Для каждой из них можно вычислить различные характеристики алгоритма $a_i^k(x) = [a(x) = k]$.

При микро-усреднении сначала эти характеристики усредняются по всем классам, а затем вычисляется итоговая двухклассовая метрика - например, точность, полнота или F-мера. Например, точность будет вычисляться по формуле

$\text{precision(a,X)} = \frac{\overline{TP}}{\overline{TP}+\overline{FP}}$

где, например, $\overline{TP}$ вычисляется по формуле $\overline{TP} = \frac{1}{K}\sum_{k=1}^{K} TP_k$

При макро-усреднении сначала вычисляется итоговая метрика для каждого класса, а затем результаты усредняются по всем классам. Например, точность будет вычислена как

$\text{precision(a,X)} = \frac{1}{K}\sum_{k=1}^{K} precision_k(a,X)$

где $\text{precision}_k\text{(a, X)} = \frac{TP_k}{TP_k+FP_k}$

В макро-усреднении каждый класс вносит равный вклад вне зависимости от размера класса.

8 SVM. Вывод оптимизационной задачи. Теория двойственности. Сильная/слабая двойственность. Вывод двойственной задачи в SVM.

Для начала теория двойственности, которая пригодится для вывода SVM.

Рассмотрим задачу минимизации с ограничениями в виде равенств и неравенств. Обратите внимание на вид неравенств.

$$\begin{cases} f_0(x) \to \min\limits_{x \in \mathbb{R}^d} \\ f_i(x) \le 0, & i = 1, \dots m, \\ h_j(x) = 0, & j = 1, \dots p \end{cases}$$

Обозначим допустимое множество за Q:

$$Q = \{x | f_i(x) \le 0, \forall i = 1, \dots m; h_i(x) = 0, \forall i = 1, \dots p\}$$

Лагранжиан, соответствующий этой задаче:

$$L(x, \lambda, \mu) = f_0(x) + \sum_{i=1}^{m} \lambda_i \cdot f_i(x) + \sum_{j=1}^{p} \mu_j \cdot h_j(x)$$

λ, μ называются множителями Лагранжа/двойственными переменными. Везде далее $\lambda \geq 0$ (если бы это было не так, то при $f_i(x) > 0$ штраф за выход за пределы допустимой области был бы отрицательным)

Двойственная задача:

$$g(\lambda,\mu) = \inf_x L(x,\lambda,\mu)$$

Двойственная задача поможет решить исходную задачу, так как даёт нижнюю оценку на минимум в исходной задаче. Покажем это:

Возьмём любой $x^1 \in Q$. Для него верно:

$$L(x^1,\lambda,\mu) = f_0(x^1) + \sum_{i=1}^{m} \lambda_i \cdot f_i(x^1) + \sum_{j=1}^{p} \mu_j \cdot h_j(x) \leq f_0(x^1)$$

Для проверки просто подставьте ограничения для произвольной допустимой точки. Далее:

$$g(\lambda,\mu) := \inf_{x \in \mathbb{R}^d} L(x,\lambda,\mu) \leq \inf_{x \in Q} L(x,\lambda,\mu) \leq L(x^1,\lambda,\mu) \leq f_0(x^1)$$

В силу произвольности выбора x^1 можно в качестве него взять решение исходной задачи (обозначим его за x^*). А значит, используя крайние члены в цепочке неравенств, имеем:

$$g(\lambda,\mu) \leq f_0(x^*)$$

Показали, что двойственная задача даёт нижнюю оценку на минимум в исходной задаче.

Теперь найдём наилучшую из нижних оценок. Будем искать её среди $g(\lambda,\mu)$ для произвольных λ,μ. Для этого нужно решить следующую задачу, называемую двойственной:

$$\begin{cases} g(\lambda,\mu) \to \max\limits_{\lambda,\mu} \\ \lambda_i \geq 0, \qquad\qquad i = 1, \dots m, \end{cases}$$

Обозначим решение этой задачи за λ^*,μ^* Свойства решения:

- $g(\lambda^*,\mu^*) \leq f_0(x^*)$ – слабая двойственность
- $g(\lambda^*,\mu^*) = f_0(x^*)$ – сильная двойственность

Для того, чтобы проверить выполняется ли сильная двойственность можно использовать достаточное условие Слейтера. Оно формулируется для выпуклой задачи оптимизации (такой, что целевая функция $f_0(x)$ и все функции в неравенствах $f_i(x), i = 1, \dots, m$ являются выпуклыми). Итак, для выпуклой задачи оптимизации выполнена сильная двойственность, если $\exists\, x^0 : f_i(x^0) < 0, i = 1, \dots,m; h_j(x^0) = 0, j = 1, \dots,p$

Если сильная двойственность выполнена, то в цепочке

$$g(\lambda^*, \mu^*) = \inf_x (f_0(x) + \sum_{i=1}^{m} \lambda_i^* \cdot f_i(x) + \sum_{j=1}^{p} \mu_j^* \cdot h_j(x)) \leq$$

$$f_0(x^*) + \sum_{i=1}^{m} \lambda_i^* \cdot f_i(x^*) + \sum_{j=1}^{p} \mu_j^* \cdot h_j(x^*) \leq f_0(x^*)$$

можно заменить все неравенства на равенства. Отсюда можно сделать следующие выводы:

1. $\sum_{i=1}^{m} \lambda_i^* \cdot f_i(x^*) = 0 \to \lambda_i^* \cdot f_i(x^*) = 0, i = 1, \ldots, m$ – условия дополняющей нежесткости (переход сделан так как каждое слагаемое неположительно)

2. Решение исходной задачи эквивалентно минимизации $L(x, \lambda^*, \mu^*)$. А значит найдя λ^*, μ^* мы сможем легко решить прямую задачу.

Теперь сформулируем теорему Каруша-Куна-Такера, которая определяет необходимые условия экстремума для прямой задачи. [**Каруша-Куна-Такера (ККТ)**] Пусть x^* является решением задачи:

$$\begin{cases} f_0(x) \to \min\limits_{x \in \mathbb{R}^d} \\ f_i(x) \leq 0, & i = 1, \ldots m, \\ h_j(x) = 0, & j = 1, \ldots p \end{cases}$$

Тогда найдутся такие λ^*, ν^*, что выполняются **условия Каруша-Куна-Такера**:

$$\begin{cases} \nabla_x L(x^*, \lambda^*, \mu^*) = 0 & \text{(условие стационарности)} \\ \lambda_i^* f_i(x^*) = 0, \, i = 1, \ldots, m & \text{(условия дополняющей нежесткости)} \\ \lambda_i^* \geq 0, \, i = 1, \ldots, m & \text{(условия неотрицательности)} \end{cases}$$

Достаточное условие:
Если прямая задача является выпуклой и выполнено условие Слейтера, то выполнение условий Каруша-Куна-Такера для (x^*, λ^*, μ^*) является достаточным и x^* – точка глобального минимума

Порядок решения задач с ограничениями в виде равенств и неравенств:

1. Выписываем Лагранжиан для прямой задачи

2. Выписываем условия Каруша-Куна-Такера

3. Перебираем возможные варианты (разные случаи выполнения условий дополняющей нежесткости), проверяем, чтобы остальные условия продолжали выполняться

4. Как только все условия выполнены останавливаем перебор и получаем точку глобального минимума.

Пример использования будет в выводе SVM. Более простые примеры смотрите в семинаре и у Соколова. Теперь к самому методу.

SVM (Support-vector machine) или метод опорных векторов – это один из методов линейной классификации. Соответственно, будем решать задачу поиска оптимальной гиперплоскости, разделяющей выборку. Напоминаю, что линейный классификатор имеет вид: $a(x) = sign(\langle w,x\rangle + b)$. Классов у нас будет два: $a(x_i) \in \{-1,+1\}$

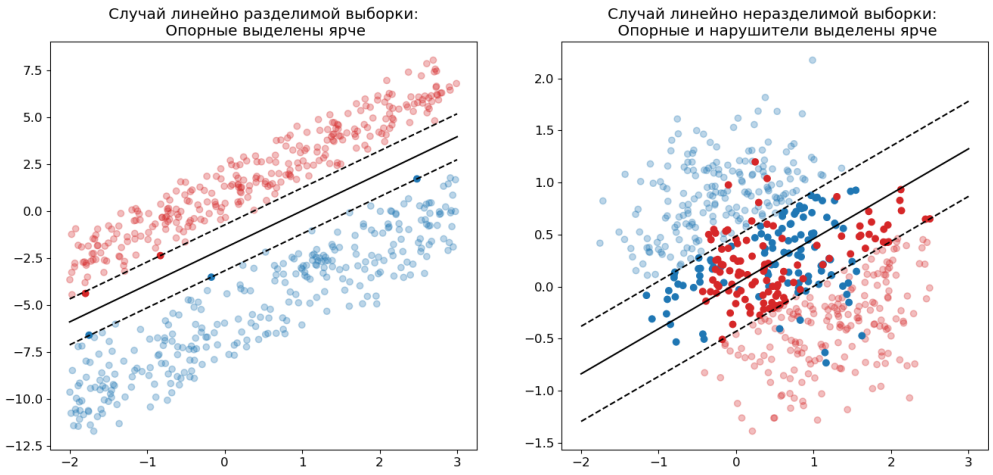


Рис. 4. Примеры классификации с помощью SVM

Сначала поговорим про случай линейно разделимой выборки, то есть такой, что можно подобрать линейный классификатор не допускающий ошибок. Идея SVM в том, чтобы подобрать классификатор, обладающей наилучшей обобщающей способностью. Интуитивно понятно, что чем больше ширина разделяющей полосы, тем увереннее классификация. Тогда потребуем, чтобы разделяющая гиперплоскость максимально далеко отстояла от ближайших к ней точек обоих классов.

Каждый классификатор имеет бесконечное множество представлений, так как он не изменится при умножении на положительную константу ($\langle w,x\rangle + b = 0$ и $const\langle w,x\rangle + const \cdot b = 0$ одно и то же. Поэтому можно ввести удобную нормировку на параметры:

$$\min_{x \in X^n} |\langle w,x\rangle + b| = 1$$

Расстояние от точки до классификатора, задаваемого гиперплоскостью вычисляется так:

$$\rho(x,a) = \frac{|\langle w,x\rangle + b|}{\|w\|}$$

Тогда расстояние от разделяющей гиперплоскости до ближайшего объекта обучающей выборки:

$$\min_{x \in X^n} \frac{|\langle w,x\rangle + b|}{\|w\|} = \frac{1}{\|w\|}$$

Значит ширина разделяющей полосы равна $\frac{2}{\|w\|}$

Тогда будем решать задачу максимизации ширины полосы при условии идеальной классификации. Воспользуемся тем фактом, что при правильной классификации $y_i(\langle w,x_i\rangle + b) \geq 0$, а так как у нас $\min\limits_{x \in X^n} |\langle w,x\rangle + b| = 1$, то в итоге имеем такую задачу:

$$\begin{cases} \frac{1}{2}\|w\|^2 \to \min\limits_{w,b} \\ y_i(\langle w,x_i\rangle + b) \geq 1, \quad i = 1,\ldots,n \end{cases}$$

Заметим, что целевая функция выпукла, а ограничения линейны (и сама задача выпукла). Следовательно, решение существует и единственно.

Будем решать с помощью двойственной задачи:

1. Лагранжиан:
$$L = \frac{1}{2}w^T w - \sum_{i=1}^{n} \lambda_i[y_i(\langle w,x_i\rangle + b) - 1]$$

2. Условия ККТ:
$$\begin{cases} \nabla_w L = w - \sum\limits_{i=1}^{n} \lambda_i \cdot y_i \cdot x_i = 0 \\ \frac{\partial L}{\partial b} = -\sum\limits_{i=1}^{n} \lambda_i \cdot y_i = 0 \\ \lambda_i[y_i(\langle w,x_i\rangle + b) - 1] = 0, \qquad i = 1,\ldots,n \\ \lambda_i \geq 0, \qquad i = 1,\ldots,n \end{cases}$$

3. Решение прямой задачи:
$$w^* = \sum_{i=1}^{n} \lambda_i y_i x_i$$

4. Составим двойственную задачу. Сначала подставляем w^* в Лагранжиан:

$$\mathrm{g}(\lambda) = \tfrac{1}{2}\langle\sum_{i=1}^{n} \lambda_i y_i x_i, \sum_{j=1}^{n} \lambda_j y_j x_j\rangle - \sum_{i=1}^{n} \lambda_i[y_i(\langle w^*,x_i\rangle + b) - 1] = \{\langle\sum_{i=1}^{n} \lambda_i y_i x_i, \sum_{j=1}^{n} \lambda_j y_j x_j\rangle =$$
$$\langle w^*, \sum_{i=1}^{n} \lambda_i y_i x_i\rangle = \sum_i \lambda_i y_i \langle w^*, x_i\rangle\} = \tfrac{1}{2}\sum_i \lambda_i y_i \langle w^*, x_i\rangle - \sum_i \lambda_i y_i \langle w^*, x_i\rangle - b\sum_i \lambda_i y_i + \sum_i \lambda_i =$$
$$-\tfrac{1}{2}\sum_i \lambda_i y_i \langle w^*, x_i\rangle - \underbrace{b\sum_i \lambda_i y_i}_{=0 \text{ ККТ } 2} + \sum_i \lambda_i = -\tfrac{1}{2}\sum_i \lambda_i y_i \langle w^*, x_i\rangle + \sum_i \lambda_i = -\tfrac{1}{2}\sum_j \sum_i \lambda_i \lambda_j y_i y_j \langle x_i,x_j\rangle +$$
$$\sum_i \lambda_i$$

Двойственная задача:

$$\begin{cases} g(\lambda) = -\frac{1}{2}\sum\limits_j \sum\limits_i \lambda_i \lambda_j y_i y_j \langle x_i,x_j\rangle + \sum\limits_i \lambda_i \to \max\limits_{\lambda} \\ \sum\limits_{i=1}^{n} \lambda_i \cdot y_i = 0 \\ \lambda_i \geq 0, \qquad i = 1,\ldots,n \end{cases}$$

Это задача квадратичного программирования, не имеющая аналитического решения. Однако, она вогнутая с аффинными ограничениями, а значит максимум у неё точно есть и он единственный. Есть алгоритмы решения на компутере, но мы их не рассматривали.

Почему SVM так называется?

Из условий дополняющей нежесткости для исходной задачи $\lambda_i > 0$ или $\lambda_i = 0$:

- Если $\lambda_i = 0$, то i-ое наблюдение не влияет на парметры классификатора ($w^* = \sum\limits_{i=1}^{n} \lambda_i y_i x_i$), такие объекты ($x_i$) называются **периферийными**
- $\lambda_i > 0$, то из дополняющей нежесткости следует, что: $y_i(\langle w,x_i\rangle + b) = 1$. Соответствующие объекты лежат на границах разделяющей полосы (на первой картинке они выделены ярче). Собственно только эти объекты и влияют на итоговые параметры классификатора, а потому называются **опорными** (и если бы мы выкинули все остальные объекты параметры бы не изменились)

Следующий логичный шаг попробовать решить аналогичную задачу (теперь она называется soft-margin SVM), но в случае линейно неразделимой выборки (2 картинка). В таком случае всегда найдётся хотя бы одно ограничение в исходной задаче, которое будет нарушаться $y_i(\langle w,x_i\rangle + b) < 1$ Вместо жёстких ограничений будем использовать мягкие и будем штрафовать за ошибки классификации и за попадание внутрь разделяющей полосы. Теперь ограничения выглядят так:

$$y_i(\langle w,x_i\rangle + b) \geq 1 - \xi_i,\ \xi_i \geq 0 - \text{ штраф}$$

Новая оптимизационная задача:

$$\begin{cases} \frac{1}{2}\|w\|^2 + C\sum\limits_{i=1}^{n} \xi_i \to \min\limits_{w,b,\xi} \\ y_i(\langle w,x_i\rangle + b) \geq 1 - \xi_i, & i = 1,\ldots,n \\ \xi_i \geq 0, & i = 1,\ldots,n \end{cases}$$

Без $C\sum\limits_{i=1}^{n} \xi_i$ веса просто занулились бы. Задача выпуклая, имеет единственный глобальный минимум.

Снова будем решать с помощью двойственности

1. Лагранжиан:

$$L = \frac{1}{2}w^T w + C\sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \lambda_i[y_i(\langle w,x_i\rangle + b) - 1 + \xi_i] - \sum_{i=1}^{n} \mu_i \xi_i$$

2. Условия ККТ:

$$\begin{cases} \nabla_w L = w - \sum\limits_{i=1}^{n} \lambda_i \cdot y_i \cdot x_i = 0 \\ \frac{\partial L}{\partial b} = -\sum\limits_{i=1}^{n} \lambda_i \cdot y_i = 0 \\ \frac{\partial L}{\partial \xi_i} = C - \lambda_i - \mu_i = 0 \\ \lambda_i[y_i(\langle w,x_i\rangle + b) - 1 + \xi_i] = 0, & i = 1,\ldots,n \\ \mu_i \xi_i = 0, & i = 1,\ldots,n \\ \lambda_i \geq 0, \xi_i \geq 0, \mu_i \geq 0, & i = 1,\ldots,n \end{cases}$$

3. Решение прямой задачи:

$$w^* = \sum_{i=1}^{n} \lambda_i y_i x_i$$

4. Составим двойственную задачу. Сначала подставляем w^* в Лагранжиан:

$$\mathrm{g}(\lambda) = \tfrac{1}{2}\langle \sum_{i=1}^{n} \lambda_i y_i x_i, \sum_{j=1}^{n} \lambda_j y_j x_j \rangle - \sum_{i=1}^{n} \lambda_i y_i (\langle w^*, x_i \rangle) - b \sum_{i=1}^{n} \lambda_i y_i + \sum_{i=1}^{n} \lambda_i + \sum_{i=1}^{n} \xi_i (C - \lambda_i - \mu_i) =$$

$$\{\langle \sum_{i=1}^{n} \lambda_i y_i x_i, \sum_{j=1}^{n} \lambda_j y_j x_j \rangle = \langle w^*, \sum_{i=1}^{n} \lambda_i y_i x_i \rangle = \sum_i \lambda_i y_i \langle w^*, x_i \rangle\} = -\tfrac{1}{2} \sum_{i=1}^{n} \lambda_i y_i \langle w^*, x_i \rangle - \underbrace{b \sum_{i=1}^{n} \lambda_i y_i}_{=0 \text{ ККТ}} + \sum_{i=1}^{n} \lambda_i +$$

$$\underbrace{\sum_{i=1}^{n} \xi_i (C - \lambda_i - \mu_i)}_{=0 \text{ ККТ}} = -\tfrac{1}{2} \sum_j \sum_i \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle + \sum_i \lambda_i$$

Двойственная задача:

$$\begin{cases} g(\lambda) = -\tfrac{1}{2} \sum\limits_j \sum\limits_i \lambda_i \lambda_j y_i y_j \langle x_i, x_j \rangle + \sum\limits_i \lambda_i \to \max\limits_{\lambda} \\ \sum\limits_{i=1}^{n} \lambda_i \cdot y_i = 0 \\ 0 \leq \lambda_i \leq C,\ i = 1,\ldots,n \end{cases}$$

Ограничение $0 \leq \lambda_i \leq C$, $i = 1,\ldots,n$ получается объединением условий ККТ ($C - \lambda_i - \mu_i = 0, \lambda_i \geq 0, \mu_i \geq 0, i = 1,\ldots,n$)

Это задача квадратичного программирования, не имеющая аналитического решения. Однако, она вогнутая с аффинными ограничениями, а значит максимум у неё точно есть и он единственный.

Снова рассмотрим на какие категории разбиваются все объекты

1. Если $\xi_i = 0, \lambda_i = 0$, то объект правильно классифицируется, лежит за пределами разделяющей полосы и не влияет на решение. Такие объекты называются периферийными.

2. Если $\xi_i = 0, 0 < \lambda_i < C$, то объект по условию дополняющей нежесткости лежит в точности на границе разделяющей полосы и влияет на итоговые параметры классификатора. Такие объекты называются опорными периферийными.

3. Если $\xi_i > 0, \lambda_i = C$, то объект лежит либо внутри разделяющей полосы($0 < \xi_i < 1$), либо за её пределами (с неправильной стороны $\xi_i \geq 2$) и влияет на решение Такие объекты называются опорными нарушителями.

Прямую задачу для soft-margin SVM можно свести к безусловной

$$\begin{cases} \frac{1}{2}\|w\|^2 + C\sum\limits_{i=1}^{n}\xi_i \to \min\limits_{w,b,\xi} \\ y_i(\langle w,x_i\rangle + b) \geq 1 - \xi_i, & i = 1,\ldots,n \\ \xi_i \geq 0, & i = 1,\ldots,n \end{cases}$$

Перепишем ограничения так:

$$\begin{cases} \xi_i \geq 1 - y_i(\langle w,x_i\rangle + b), & i = 1,\ldots,n \\ \xi_i \geq 0, & i = 1,\ldots,n \end{cases}$$

Можно объединить эти ограничения в одно:

$$\xi_i = \max(0,1 - y_i(\langle w,x_i\rangle + b))$$

Тогда можно выписать задачу безусловной оптимизации:

$$\underbrace{\frac{1}{2}\|w\|^2}_{l_2\text{-регуляризация}} + \underbrace{C\sum_{i=1}^{n}\max(0,1 - y_i(\langle w,x_i\rangle + b))}_{\text{верхняя оценка для margin}} \to \min_{w,b}$$

Такая верхняя оценка для margin: $\max(0, 1 - M)$ называется hinge-loss. То есть по сути SVM пытается найти наилучшую верхнюю оценку для ступенчатого функционала $[M_i < 0]$ с дополнительной регуляризацией.

9 Ядровая функция. Построения ядер, примеры ядер, теорема Мерсера, обобщения линейных методов с помощью ядер.

9.1 Зачем нужны

Ядра - это подход к изменению признакового пространства. Применяя ядра, можно повышать размерность пространства без вычислительных трудностей (благодаря

тому, что ядро выражается как скалярное произведение признаков из исходного пространства).

Если перейти к новым признакам с помощью ядер, то можно восстановить нелинейные зависимости в данных, используя только линейные функции. Ядро поможет линейно разделить выборку, которая по исходным признакам не была линейно разделимой.

9.2 Определение

Ядром называется функция $K(x,z)$, представимая в виде $K(x,z) =< \gamma(x),\gamma(z) >$, где $\gamma : X \to H$ (γ - базисная функция, H - спрямляющее пространство)

9.3 Теорема Мерсера

$K(x,z)$ является ядром, если:

1) Симметричность: $K(x,z) = K(z,x)$

2) Неотрицательная определённость: $K(x,z)$ неотрицательно определена, то есть $\forall$ конечной выборки, $K(x_i,x_j)_{i,j=1}^{n}$ неотрицательно определена.

9.4 Свойства ядер (используются для их построения)

Положим $\alpha > 0, K_1, K_2$ - ядра. Тогда:

1) $K_1 + K_2$ - ядро
2) $K_1 * K_2$ - ядро
3) $\alpha * K_1$ - ядро
4) $f(x) * f(z)$ - ядро (f - некоторая вещественная функция)
5) $K(\gamma(x),\gamma(z))$ - ядро
6) $\lim_{n\to\infty} K_n(x,z)$ - ядро

9.5 Примеры ядер и их построения

1) Полиномиальное.

Если p(v) - многочлен с положительными коэффициентами (св-во 3), то $K(x,z) = p(< x,z >)$ - ядро.

Частный случай такого ядра:

$$K(x,z) = (< x,z > +R)^m$$

2) Гауссовское (применяется чаще всего).

$$K(x,z) = exp(\frac{-||x-z||^2}{2\sigma^2})$$

Ему соответствует бесконечномерное спрямляющее пространство (доказательство через разложение экспоненты в ряд Тейлора).

! Чем меньше σ, тем больше вес при мономах большой степени, тем больше риск переобучения.

3) RBF. Теоретически это $K(x,z) = K(||x-z||)$, на практике в sk-learn так называют гауссовское ядро.

9.6 Обобщение линейных методов с помощью ядер

Из документа на он.эконе "Соколов. Ядра 1 параграфы 1.1-1.2.

Будем искать зависимость как линейную комбинацию нелинейных функций от выборки:

$$y(x) = \sum_{i=1}^{m} w_i \gamma_i(x)$$

Функционалом возьмём MSE, добавим регуляризацию:

$$Q(w) = \frac{1}{2}\sum_{i=1}^{n}(\sum_{j=1}^{m} w_i \gamma_i(x) - y)^2 + \frac{\lambda}{2}||w||^2 \to min_w$$

.

Перепишем в матричном виде (Ψ - матрица, в которой i-я строка является вектором $[\gamma_1(xi), ..., \gamma_m(xi)]$):

$$Q(w) = \frac{1}{2}||\Psi - y||^2 + \frac{\lambda}{2}||w||^2 \to min_w$$

Дифференцируем функционал $Q(w)$, приравниваем нулю градиент. Находим

$$w = -\frac{1}{\lambda}\Psi^T(\Psi w - y) = \Psi^T * a$$

где за a обозначили вектор $-\frac{1}{\lambda}(\Psi w - y)$. а ещё надо будет найти, там остались неизвестные λ.

Подставим решение в функционал:

$$Q(a) = \frac{1}{2}||\Psi\Psi^l a - y||^2 + \frac{\lambda}{2}a^T\Psi\Psi^T a \to min_a$$

$K = \Psi\Psi^T$ - это матрица Грамма, то есть матрица скалярных произведений всех возможных пар объектов. Её элемент ij расписывается как $k(x_i,x_j) =< \gamma(x_i),\gamma(x_j) >$ - это ядро.

Оптимальный вектор a в результате вычислений получится $a = (K + \lambda I)^{-1}y$
Искомая функция регрессии тогда

$$y(x) = \sum_{i=1}^{m} w_i \gamma_i(x) = w^T\gamma(x) = a^T\Psi\gamma(x) = k(x)^T(K + \lambda I)^{-1}y$$

где $k(x) = [k(x, x_1), ..., k(x, x_n)]$ - вектор скалярных произведений нового объекта x на объекты обучающей выборки $x_1, ..., x_n$.

Результат - переписали функционал и модель так, что они зависят лишь от скалярных произведений объектов.

Оптимальность применения ядер здесь в том, что (как доказали на лекции) для вычисления таких скалярных произведений надо всего лишь столько операций, сколько исходно объектов в выборке.

10 Задача снижения размерности. Вывод постановки оптимизационной задачи PCA, её решение. Критерии отбора оптимального числа главных компонент.

10.1 Задача снижения размерности

Целью задачи снижения размерности является представление объекта $x \in R^D$ в пространстве $\hat{x} \in R^d$, где $D > d$. Пусть есть некоторое линейное подпространство $L = L\{\vec{a_1}, \vec{a_2}, \cdots, \vec{a_d}\}$ в пространстве R^D (если у нас дано многообразие, то можно его свести к подпространству, отцентрировав, вычтя из данных среднее ($\overline{x}$)). Пусть матрица A равна векторам подпространства L, выписанным по столбцам:

$$A = \underbrace{\begin{bmatrix} \vdots & \vdots & \vdots \\ \vec{a_1} & \vdots & \vec{a_n} \\ \vdots & \vdots & \vdots \end{bmatrix}}_{Dxd}$$

Вектор принадлежит подпространству, если выражается через нетривиальную линейную комбинацию векторов этого подпространства: $\vec{x} = \alpha_1\vec{a_1} + \cdots + \alpha_d\vec{a_d}$, в векторно-матричном виде:

$$\underbrace{\vec{x}}_{Dx1} = A\alpha = \underbrace{\begin{bmatrix} \vdots & \vdots & \vdots \\ \vec{a_1} & \vdots & \vec{a_n} \\ \vdots & \vdots & \vdots \end{bmatrix}}_{Dxd} * \underbrace{\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_d \end{bmatrix}}_{dx1} = \begin{bmatrix} \vdots \\ \vec{a_1} \\ \vdots \end{bmatrix} \alpha_1 + \cdots + \begin{bmatrix} \vdots \\ \vec{a_d} \\ \vdots \end{bmatrix} \alpha_d$$

Таким образом, x в базисе L будут коэффициенты вектора α (маломерное представление x):

$$\alpha_x = A^T x$$

Комментарий из ноутбука Андрейцева к рисунку:В примере выше исходная точка x=(1,1.5,10.5). Если в качестве базиса плоскости принять вектора a_1=(0,1,5), a_2=(1,0,3), то координаты вектора x в базисе этой плоскости принимают вид x_a=(1.5,1)

Задача снижения размерности имеет вид:

$$\begin{cases} \frac{1}{2}\|x - x_0\|^2 \to \min\limits_x \\ x = A\alpha \end{cases}$$

Подставим x и получим:

$$Q = \frac{1}{2}\|A\alpha - x_0\|^2 \to \min_\alpha$$

$$= \frac{1}{2}(\alpha^T A^T A\alpha - 2x_0^T A\alpha + x_0^T x_0) \to \min_\alpha$$

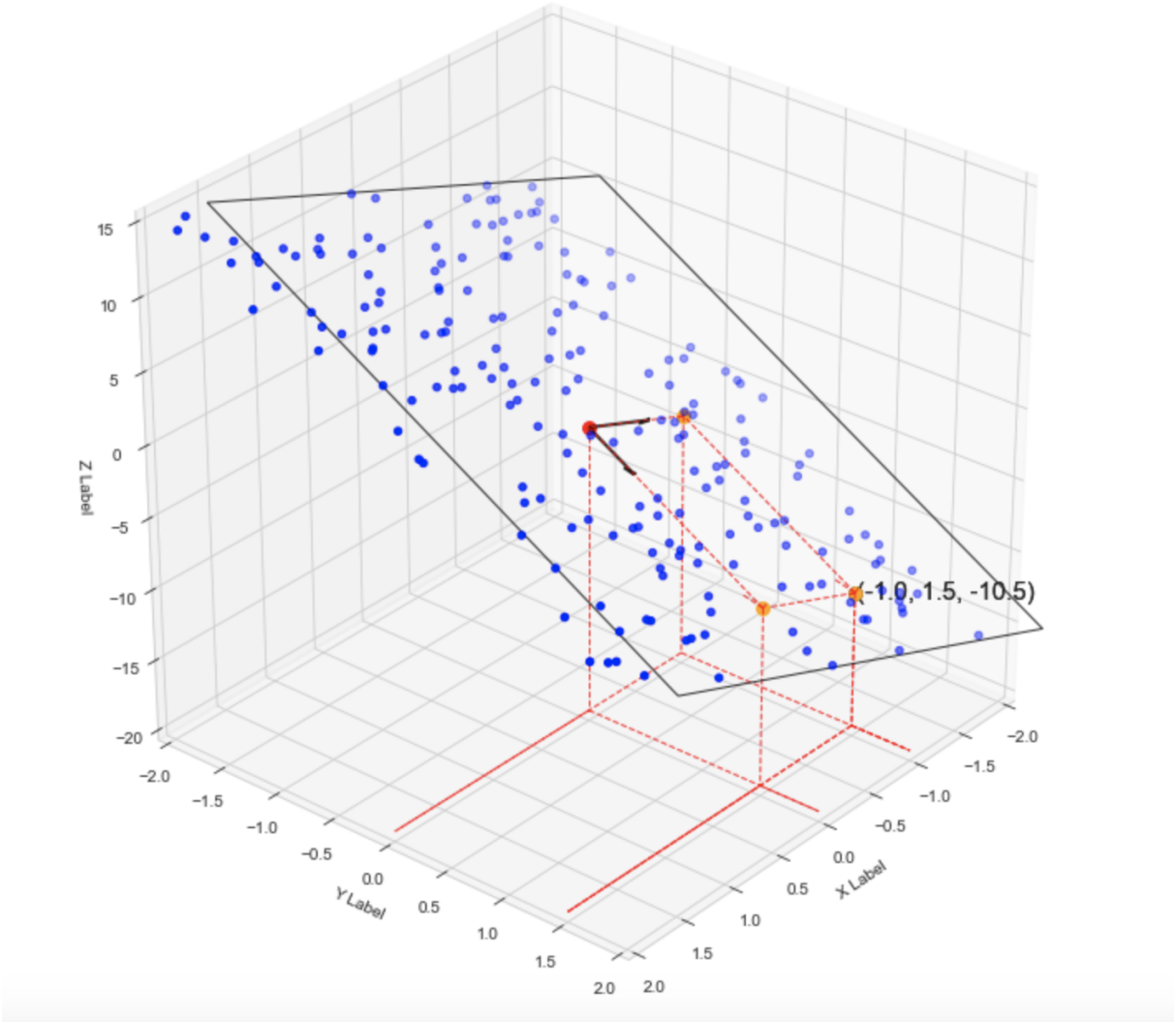


Рис. 5. Проекция из трёхмерного пространства в двухмерное

$$\nabla_\alpha Q = A^T A\alpha - A^T x_0 = 0 \to \alpha = (A^T A)^{-1} A^T x_0$$

Любой базис через процесс Грама-Шмидта можно ортонормировать, поэтому изначально ортонормируем L, чтобы $A^T A = I$. (AA^T не всегда равно I, только когда матрица A — квадратная, то есть количество признаков равно количеству наблюдений) Итого:

$$\alpha = A^T x_0$$

Проекция x_0 на L:

$$proj_L x_o = AA^T x_0$$

10.2 PCA

Первым делом для использования алгоритма нужно отцентрировать данные (то, о чём говорилось в предыдущей части – вычесть среднее значение). В пакетах обычно зашивают стандартизацию, то есть при этом каждое наблюдение ещё и делится на

стандартное отклонение признака. Ещё на лекции говорили, что есть альтернатива: загнать наблюдения в отрезок от 0 до 1 (вычесть минимум по выборке и поделить на разницу между максимума и минимума). Какой из этих методов лучше не всегда понятно, но всегда можно проверить на практике.

Классическая стандартизация: $x_i = \frac{x_i - \overline{x}}{\sigma_x}$

Альтернатива: $x_i = \frac{x_i - x_{min}}{x_{max} - x_{min}}$

Мы хотим найти минимальное среднее расстояние расстояние от каждого x_i до плоскости L:

$$Q = \frac{1}{n}\sum_{i=1}^{n} \rho(x_i; L) \to \min_{L}$$

То есть расстояние до проекции:

$$\frac{1}{n}\sum_{i=1}^{n} \|x_i - AA^T x_i\|^2 \to \min_{A=a_1,\cdots,a_d}$$

Такая оптимизационная задача нам дает два преимущества:

1. Мы можем выбрать расположение подпространства
2. Мы можем выбрать число векторов d, по которым будет происходить минимизация

Преобразуем:

$$\frac{1}{n}\sum_{i=1}^{n}(x_i^T - x_{iT}AA^T)(x_i - AA^T x_i) = \frac{1}{n}\sum_{i=1}^{n} x_i^T(I - AA^T)(I - AA^T)x_i = \frac{1}{n}\sum_{i=1}^{n} x_i^T(I - AA^T)x_i \to \min_{A=a_1,\cdots,a_d}$$

Объяснение последнего перехода (матрицы, для которых это верно называются идемпотентными): $(I - AA^T)(I - AA^T) = I - AA^T - AA^T + A\underbrace{A^TA}_{I}A^T = I - 2AA^T + AA^T = I - AA^T$

Раскладываем:

$$\frac{1}{n}\sum_{i=1}^{n} x_i^T x_i - \frac{1}{n}\sum_{i=1}^{n} \underbrace{\underbrace{x_i^T A}_{(A^T x_i)^T} A^T x_i}_{\|A^T x_i\|^2} \to \min_{A}$$

Первое слагаемое не зависит от A, а у второго уберем минус и перепишем задачу на максимум:

$$Q = \frac{1}{n}\sum_{i=1}^{n} \|A^T x_i\|^2 \to \max_{A}$$

$$A^T x_i = \underbrace{\begin{bmatrix} \cdots & \vec{a_1} & \cdots \\ \vdots & \vdots & \vdots \\ \cdots & \vec{a_d} & \cdots \end{bmatrix}}_{dxD} * \underbrace{\begin{bmatrix} \alpha_1 \\ \vdots \\ \alpha_d \end{bmatrix}}_{Dx1} = \begin{bmatrix} \langle a_1, x_i \rangle \\ \vdots \\ \langle a_d, x_i \rangle \end{bmatrix}$$

$\|A^T x_i\|^2 = \langle a_1, x_i\rangle^2 + \cdots + \langle a_d, x_i\rangle^2 = \sum\limits_{k=1}^{d}\langle a_k, x_i\rangle^2$ Подставляем в Q:

$$Q = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{d}\langle a_k, x_i\rangle^2 \to \max_{\vec{a_1},\cdots,\vec{a_d}}$$

$$\langle a_k, x_i\rangle^2 = \langle a_k, x_i\rangle\langle a_k, x_i\rangle = a_k^T x_i * x_i^T a_k$$

Подставляем:

$$Q = \frac{1}{n}\sum_{i=1}^{n}\sum_{k=1}^{d} a_k^T x_i * x_i^T a_k = \sum_{k=1}^{d} a_k^T \underbrace{(\frac{1}{n}\sum_{i=1}^{n} x_i x_i^T)}_{S} a_k$$

S – это выборочная ковариационная матрица признаков, так как изначально мы их центрировали и их среднее значение равно 0. Теперь выпишем задачу PCA, помня об ортонормированности (по ходу будет видно, что последние условие не нужно):

$$\begin{cases} \sum\limits_{k=1}^{d} a_k^T S a_k \to \max\limits_{\vec{a_1},\cdots,\vec{a_d}} \\ \|a_k\|^2 = 1 \\ \langle a_i, a_j\rangle = 0, \forall i \neq j \end{cases}$$

Задача распадается на d независимых подзадач, так как найти максимум сумму эквивалентно поиску максимума для каждого элемента суммы:

$$\begin{cases} a_k^T S a_k \to \max\limits_{\vec{a_k}} \\ \|a_k\|^2 = 1 \\ \langle a_i, a_j\rangle = 0, \forall i \neq j \end{cases}$$

Запишем Лагранджиан:

$$L = a_k^T S a_k + \mu(a_k^T - 1) \to \max_{\vec{a_k}}$$

Выпишем градиент:

$$\nabla_{ak} L = 2Sa_k + 2 * \mu a_k = 0$$

$$Sa_k = -\mu a_k \to Sa_k = \gamma a_k$$

Заметим, что a_k является собственным вектором матрицы S, тогда задача сводится к поиску максимального собственного значения.

$$a_k^T S a_k = a_k^T \gamma a_k = \gamma \underbrace{a_k^T a_k}_{1} = \gamma \to \max$$

S — самосопряженный оператор, поэтому все его собственные значения вещественные, а собственные вектора, соответствующие различным собственным значениям, ортогональны и линейно независимы. Отнормировать любой вектор не проблема. Итого получаем автоматически ортонормированные вектора. Еще все собственные значения положительные (откуда это берется, Андрейцев объяснить не смог). Если получатся n одинаковых собственных значений, что на практике практически невозможно, то их можно ортагонализовать процессом Грама-Шмидта.

10.3 Критерии отбора оптимального числа главных компонент

Сортируем все собственные значения от большего к меньшему.

1. Строим картинку, где по x — номер собственного значения, по y — величина собственного значения (можно поделённую на сумму всех значений). Смотрим на нее. Там, где наблюдается резкое уменьшение абсолютной величины собственного значения, то дальше мы значения мы не берем (на картинке 1 последнее значение d=2).

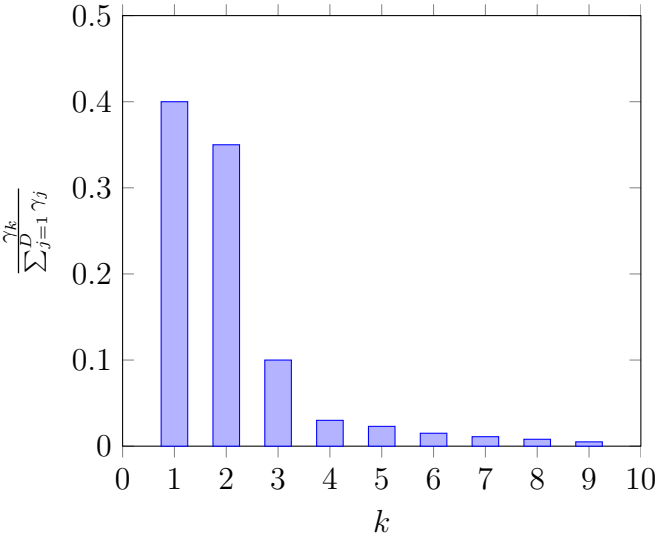
2. Можно нарисовать картинку, где по оси x будут также номера собственных значений, а по y — их накопленная доля от суммы всех собственных значений. (картинка 2). Когда эта доля станет нас устраивать, например, 3 признака объясняют 80 процентов дисперсии, то тогда отбираем соответствующее количество признаков.

3. Если нужно визуализировать данные, то 2 или 3 признака.

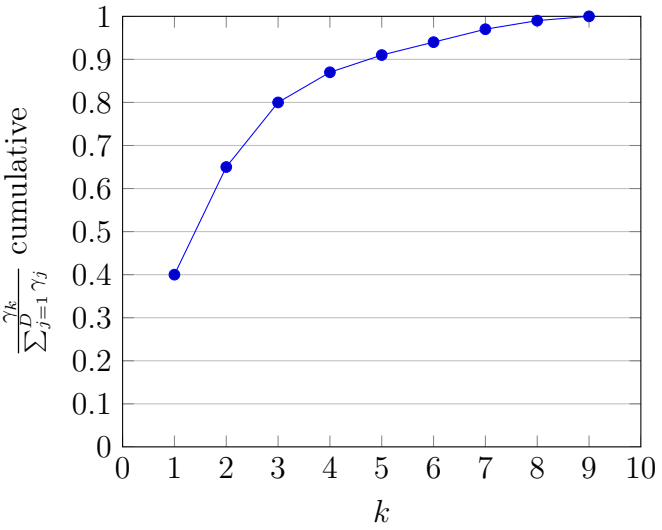
4. Если есть исходное знание о данных. Например, есть биржевой индекс, который зависит в основном от курса акций трех компаний, тогда и новая размерность будет 3.

5. Собственно, если мы используем PCA в какой-либо задаче/модели, то число главных компонент можно рассматривать как обычный гиперпараметр, и искать такое значение, при котором используемая нами метрика качества принимает максимальное значение.

Картинка 1



Картинка 2


$\frac{\gamma_k}{\sum_{j=1}^{D}\gamma_j}$ cumulative
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0
0 1 2 3 4 5 6 7 8 9 10
k

11 Word2Vec. SkipGram NS и CBOW. Применение для задач, связанных с обработкой текстов.

Word embedding (векторное представление слов) - сопоставление произвольному объекту (слову) некоторого числового вектора в пространстве фиксированной размерности.

1. Самый простой способ - **one-hot Vectors**, когда i-тому слову из *словаря* (список уникальных слов) ставится в соответствие 1 в i-той колонке, в остальных колонках - 0. (как бинарная переменная). Проблемы этого метода заключаются в следующем:

   (a) Большая размерность (если очень много уникальных слов, очень широкая матрица)

   (b) Такой способ embedding'a не "ловит"смысл слов: "расстояние"между векторами, соответствующими словам "собака "кошка"и "стол могут быть одинаковыми.

   Как решить последнюю проблему? -> учитывать контекст

2. **Word2Vec**
   Цель: учитывать в контекст в векторном представлении слов.
   Дистрибутивная гипотеза: слова, встречающиеся в похожих контекстах, имеют схожий смысл.
   Работа алгоритма: для каждой позиции в тексте t = 1, .., T алгоритм предсказывает контексные слова, на основе данных о ширине окна m (количество слов слева и справа от центрального) и самом центральном слове w_t (тогда соответственно w_{t+j}-слово внутри окна):

$$Likelyhood = L(\theta) = \prod_{t=1}^{T} \prod_{-m \le j \le m, j \neq 0} P(w_{t+j}|w_t, \theta)$$

$$Loss = -\frac{1}{T} logL(\theta) = -\frac{1}{T} \sum_{t=1}^{T} \sum_{-m \le j \le m, j \neq 0} logP(w_{t+j}|w_t, \theta) \to \min_{\theta}$$

   Таким образом, в функции потерь заложено, чтобы алгоритм пробегался по всему тексту и подсчитывал вероятности внутри окна. Как считать эти вероятности?

$$P(out|center) = \frac{u_{out}^T v_{center}}{\sum_{w \in V} exp(u_w^T v_{center})}$$

   (P.S. похоже на softmax)
   Для каждого слова w имеем два вектора: 1) v_w - когда w является центральным словом и 2) u_w - когда слово w является контекстом. То есть создаются две матрицы V и U размерностью объем словаря(всего уникальных слов) x h (гиперпараметр). Итого: наш обучаемый параметр θ - это матрицы U, V. Обучение происходит методом градиентного спуска $\theta^{new} = \theta^{old} - \alpha \nabla_\theta J(\theta)$.

Оптимизация происходит последовательно для пары слово из центра-одно слово из контекста. В результате 1 шага оптимизации обновляется 1 строка матрицы V и все строки матрицы U. Пример:

$$Loss = J_{t,j}(\theta) = -logP(cute|cat) = -log\frac{expu_{cute}^T v_{cat}}{\sum_{w \in Voc} expu_w^T v_{cat}} = -u_{cute}^T v_{cat} + log \sum_{w \in Voc} expu_w^T v_{cat}$$

Настройка происходит таким образом, что схожесть между u_{cute}^T, v_{cat} возрастает, а между v_{cat} и остальными векторами из матрицы-контекста (кроме cute) снижается.

3. **Skip-Gram Negative Sampling**
   Проблема подхода, описанного выше, состоит в том, что на каждом шаге оптимизации (то есть для каждой пары центр-контекст) происходит обновление |V|+1 векторов (вектор-центр v и все векторы-строки из контекстной матрицы U)
   Что изменилось? Теперь теперь на каждом шаге (для центрального слова cat и слова из окна cute) мы будем оптимизировать (обновлять) следующие параметры: улучшать схожесть между вектором-центром v_{cat} из матрицы V и вектором из контекса u_{cute} и снижать схожесть между v_{cat} и k случайными векторами $u_w, w \in K$, то есть итого обновляются K+2 вектора.

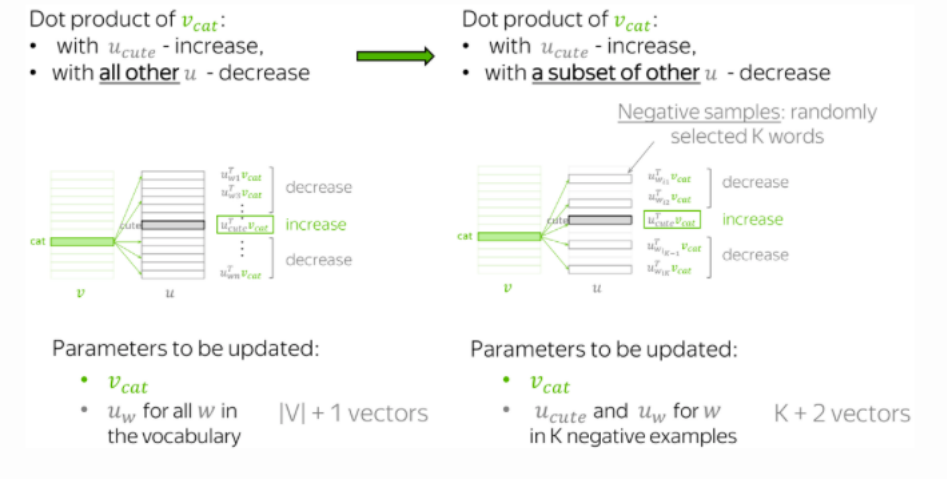


Рис. 6. negative sampling

4. **CBOW**
   Также использует k слов из матрицы-контекста, но предсказывает центральное слово по контексту, а не наоборот как Word2Vec.

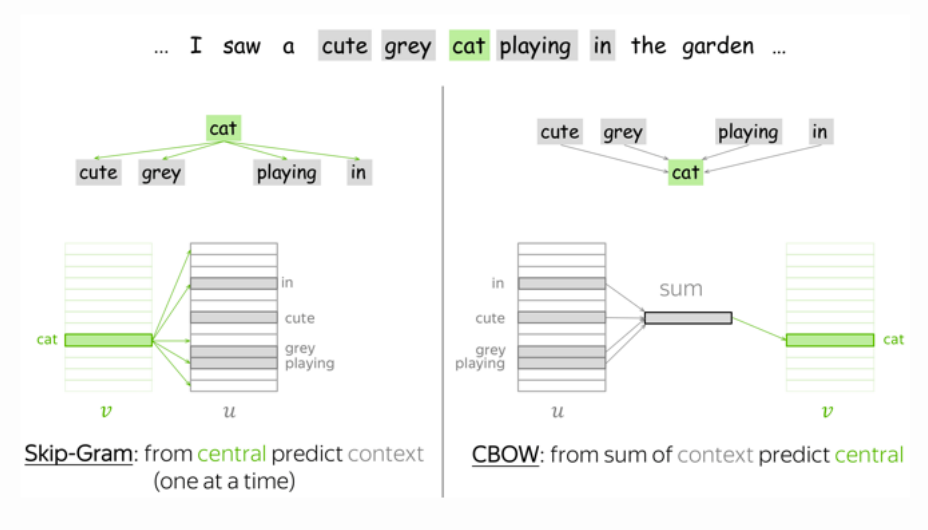


Рис. 7. Skip-Gram and CBOW

12 Постановка задачи кластеризации. Алгоритмы K-means, DBSCAN, Иерархическая кластеризация. Критерии качества кластеризации.

12.1 Постановка задачи кластеризации

Задача кластеризации (относится к типу задач «обучение без учителя»): выявить в данных K кластеров – областей, где объекты внутри одного кластера похожи друг на друга, а объекты из разных кластеров друг на друга не похожи.
Пусть есть выборка объектов X размера n. Нужно построить алгоритм a, который будет присваивать каждому элементу выборки свой кластер из {1,..,K} - $a : X \to \{1,..,K\}$.

12.2 Алгоритм k-means

Есть выборка объектов X размера n. Пусть A_k-это множества-кластеры, $A_k = \{x_i : \rho(x_i, c_k) \leqslant \rho(x_i, c_j) \forall j \neq k\}$. Центры кластеров - $\{c_1,...,c_k\}$. Хотим подобрать такие кластеры , чтобы расстояния от центров до $x_i \in A_k$ было минимально.

$$Q = \sum_{k=1}^{K} \sum_{i:x_i \in A_k} ||x_i - c_k||^2 \to \min_{c_1,...c_K}$$

Решение задачи (она распадается на K отдельных подзадач):

$$||x_i - c_k||^2 = (x_i^T - c_k^T)(x_i - c_k) = x_i^T x_i - 2x_i^T c_k + c_k^T c_k$$

Возьмем градиент для i-ого наблюдения и k-ого центра:

$$\nabla Q_{c_k} = -2x_i + 2c_k$$

Рассмотрим k - ый кластер:

$$\nabla Q_{c_k} = \nabla \sum_{i:x_i \in A_k} ||x_i - c_k||^2 = \sum_{i:x_i \in A_k} (-2x_i + 2c_k) = 0$$

$$- \sum_{i:x_i \in A_k} x_i + |A_k| \cdot c_k = 0$$

$$c_k^* = \frac{1}{A_k} \cdot \sum_{i:x_i \in A_k} x_i$$

Итог: c_k - среднее из объектов A_k.

Алгоритм состоит из следующих шагов:
0. Задаем произвольным образом центры c_1,...,c_k
Шаги 1 и 2 повторяются до сходимости алгоритма

1. На основе расстояний точек выборки до центров пересчитываем кластеры A_k
2. Задаем новые центры c_1,...,c_k по выведенной формуле c_k^*

Критерии останова (сходимости) алгоритма:

1. Можно смотреть на долю наблюдений выборки, сменивших кластер
2. Можно задать выполнение алгоритма , пока для каждого центра не выполнится следующее неравенство:
$$||c_k^{old} - c_k^{new}|| < \epsilon$$
3. Задать ограничение на функционал Q:
$$\frac{Q^{new} - Q^{old}}{Q^{old}}$$

Проблема алгоритма k-means - необходимо задать K (кол-во кластеров).

12.3 DBSCAN

DBSCAN - еще один алгоритм кластеризации (умеет работать с шумными данными). Согласно алгоритму выборка разбивается на три типа точек (красные, желтые, синие) в зависимости от параметров ϵ и $min\_samples$ - гиперпараметры. Основная идея алгоритма заключается в том, что для каждой точки **кластера** окрестность радиуса (ϵ) должна содержать как минимум $min\_samples$ точек.
алгоритм Точки делятся следующим образом:

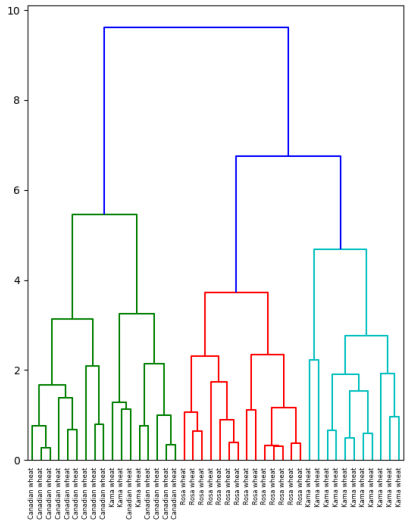
- точка помечается как «основная» , если в ее ϵ окрестности есть $min\_samples$ точек
- точка помечается как «достижимая», если в ее ϵ окрестности нет $min\_samples$ точек, но есть точки, помеченные как «основные»

- точка помечается «выбросом», если для нее не выполняются оба условия выше

Кластер образуют «основные» и «достижимые» точки (последние будут лежать на границе кластера). Если назначить ϵ слишком большим алгоритм построит 1 большой кластер. Если слишком маленьким и при этом $min\_samples = 0$ -> все точки-отдельные кластеры. Если ϵ мал , а $min\_samples > 0$ -> алгоритм определит почти все точки , как выбросы.
Преимущества алгоритма : умеет распознавать кластеры различной формы (не только сферической, как в основном делает алгоритм k-means), хорошо работает на данных, в которых есть шум.

12.4 Иерархическая кластеризация

Иерархическая кластеризация - алгоритм, который строит иерархию кластеров. Этот алгоритм начинает работу с того, что каждому экземпляру данных сопоставляется свой собственный кластер. Затем два ближайших кластера объединяются в один и так далее, пока не будет образован один общий кластер. Например, при кластеризации новостей можно рассчитывать, что чем ниже мы спускаемся по иерархии, тем более тонкие различия между сюжетами будут выделяться. На рисунке ниже представлена иллюстрация построения алгоритма. Описанный выше подход назы-



вается восходящей кластеризация. Она начинается с нижнего уровня, на котором все объекты $\{x_1,..,x_n\}$ принадлежат к отдельным кластерам:C^n (самый нижний) = $\{\underbrace{\{x_1\},}_{\text{первый кластер}} \{x_2\},...,\{x_n\}\}$. Следующие уровни получаются путем объединения наиболее похожих кластеров с предыдущего уровня.$A^j = \{X_1,..,X_j\}$, где $\{X_1,..,X_j\}$ - новые кластеры. Схожесть кластеров определяется с помощью некоторой функции $d(X_m,X_n)$ — например, это может быть расстояние между центрами кластеров.
Иерархическая кластеризация хуже подходит для кластеризации больших объемов данных в сравнении с методом k-средних (сложность иерархического алгоритма $O(n^2)$,

для k-means $O(n)$). Иерархический алгоритм менее чувствителен к зашумленным данным.

12.5 Критерии качества кластеризации

Похожесть объектов определяется метриками качества. Метрики качества (или критерии качества) показывают на сколько правильно работает алгоритм. Существует 2 вида критериев : **внутренние** и **внешние**.
Внешние требуют использование дополнительных данных (например, информацию об истинных данных). Если известно истинное распределение объектов по классам, то можно рассматривать задачу кластеризации, как задачу многоклассовой классификации, и в качестве внешних критериев использовать F-меру, например.
Внутренние метрики основаны на свойствах выборки и кластеров. Пусть c_k - центр k-ого кластера. Примеры внутренних метрик:

- 1. Внутрикластерное расстояние - суммарное расстояние между объектами внутри кластера по всем кластерам (это расстояние нужно минимизировать).

$$\sum_{k=1}^{K}\sum_{i=1}^{n}[a(x_i)=k]\rho(x_i,c_k)$$

- Межкластерное расстояние - сумма расстояний между каждой парой объектов из разных кластеров (это расстояние нужно максимизировать - объекты из разных кластеров должны быть менее похожи).

$$\sum_{i,j=1}^{n}[a(x_i)\neq a(x_j)]\rho(x_i,x_j)$$

- Индекс Данна (формула с лекции Антона) - отношение внутрикластерного и межкластерного расстояния (хотим минимизировать)

$$\text{Dunn Index}=\frac{\rho_{\text{внутрикластерное}}}{\rho_{\text{межкластерное}}}$$

- Силуэт - показывает, насколько среднее расстояние до объектов своего кластера отличается от среднего расстояния до объектов других кластеров.Силуэтом выборки называется средняя величина силуэта объектов данной выборки. Введем обозначения :
  a — среднее расстояние от данного объекта до объектов из того же кластера, b — среднее расстояние от данного объекта до объектов из ближайшего кластера (отличного от того, в котором лежит сам объект).
  Силуэт для отдельного объекта:

$$Silhouette_i=\frac{b-a}{max(a,b)}$$

  Расчитываем силуэт для каждого объекта выборки и усредняем. Данная величина лежит в диапазоне [-1,1]. Значения, близкие к -1, соответствуют плохим (разрозненным) кластеризациям, значения, близкие к нулю, говорят о том, что кластеры пересекаются и накладываются друг на друга, значения, близкие к 1, соответствуют «плотным» четко выделенным кластерам.

13 Bias-variance decomposition. Примеры разложений для некоторых моделей (например для бэггинга).

Ошибка любой модели складывается из трёх факторов: сложности самой выборки, сходства модели с истинной зависимостью ответов от объектов в выборке, и богатства семейства, из которого выбирается конкретная модель. Между этими факторами существует некоторый баланс, и уменьшение одного из них приводит к увеличению другого. Такое разложение ошибки носит название разложения на смещение и разброс.

Пусть есть какое-то распределение фичей (x) и соответствующих им ответов (y): $p(x,y)$ – *плотность распределения*. Из этого распределения генерируется выборка X и ответы на ней. Будем рассматривать задачу регрессии (y – вещественное число). Пусть a – наш алгоритм. В качестве функции потерь будем рассматривать квадратичную функцию : $L(y,a) = (y-a(x))^2$, которой соответствует среднеквадратическая ошибка: $E_{x,y}(y - a(x))^2$ (это MSE). Данный функционал усредняет ошибку модели в каждой точке пространства x и для каждого возможного ответа y, причём вклад пары (x, y), по сути, пропорционален вероятности получить её в выборке $p(x, y)$. Разумеется, на практике мы не можем вычислить данный функционал, поскольку распределение $p(x, y)$ неизвестно. Тем не менее, в теории он позволяет измерить качество модели на всех возможных объектах, а не только на обучающей выборке.

При минимизации MSE оптимальным алгоритмом будет $a = E(y|x)$. Докажем это.

13.1 Минимум среднеквадратичного риска

Преобразуем функцию потерь:

$$\begin{aligned} L(y, a(x)) &= (y - a(x))^2 = (y - E(y|x) + E(y|x) - a(x))^2 = \\ &= (y - E(y|x))^2 + 2\big(y - E(y|x)\big)\big(E(y|x) - a(x)\big) + (E(y|x) - a(x))^2. \end{aligned}$$

Подставляя её в функционал среднеквадратичного риска, получаем:

$$\begin{aligned} R(a) &= E_{x,y}L(y, a(x)) = \\ &= E_{x,y}(y - E(y|x))^2 + E_{x,y}(E(y|x) - a(x))^2 + \\ &+ 2E_{x,y}\big(y - E(y|x)\big)\big(E(y|x) - a(x)\big). \end{aligned}$$

Разберемся сначала с последним слагаемым. Перейдём от матожидания $E_{x,y}[f(x, y)]$ к цепочке матожиданий: $E_{x,y}[f(x, y)] = E_x E_y[f(x, y)|x]$ и заметим, что величина $\big(E(y|x) - a(x)\big)$ не зависит от y, и поэтому ее можно вынести за матожидание по y:

$$\begin{aligned} &E_x E_y\Big[\big(y - E(y|x)\big)\big(E(y|x) - a(x)\big)|x\Big] = \\ &= E_x\Big(\big(E(y|x) - a(x)\big)E_y\Big[\big(y - E(y|x)\big)|x\Big]\Big) = \\ &= E_x\Big(\big(E(y|x) - a(x)\big)\big(E(y|x) - E(y|x)\big)\Big) = \\ &= 0 \end{aligned}$$

Получаем, что функционал среднеквадратичного риска имеет вид

$$R(a) = E_{x,y}(y - E(y|x))^2 + E_{x,y}(E(y|x) - a(x))^2.$$

От алгоритма $a(x)$ зависит только второе слагаемое, и оно достигает своего минимума, если $a(x) = E(y|x)$. Таким образом, оптимальная модель регрессии для квадратичной функции потерь имеет вид

$$a_*(x) = E(y|x) = \int_y p(y|x)dy.$$

Иными словами, мы должны провести «взвешенное голосование» по всем возможным ответам, причем вес ответа равен его апостериорной вероятности. Ч.Т.Д.

Чтобы использовать такой алгоритм, нам необходимо знать распределение, но мы его не знаем $\Longrightarrow$ переходим к выбору алгоритма на основе **метода обучения**.

13.2 Ошибка метода обучения

Из обучающей выборки генерируем выборки (произвольным способом) с фиксированным размером n. Для каждой выборки выбираем метод обучения – μ (он выбирается из некоторого семейства алгоритмов A), обучаем μ на соответствующей ему выборке и получаем предикты:

$$a(x)_{X^n} = \mu(X^n)(x) \text{ – это уже ответы}$$

Для оценки качества **метода обучения** берём среднее по всем выборкам значение среднеквадратической ошибки (усредняем MSE по всем X^n):

$L(\mu) = E_{X^n}(E_{x,y}(y-\mu(X))^2)$ – ошибка, разложение на компоненты которой хотим понять

Выше мы показали, что среднеквадратичный риск на фиксированной выборке X можно расписать как

$$E_{x,y}\Big[\big(y - \mu(X)\big)^2\Big] = E_{x,y}\Big[\big(y - E[y|x]\big)^2\Big] + E_{x,y}\Big[\big(E[y|x] - \mu(X)\big)^2\Big].$$

Подставим это представление в $L(\mu)$:

$$\begin{aligned} L(\mu) &= E_X\Big[\underbrace{E_{x,y}\Big[\big(y - E[y|x]\big)^2\Big]}_{\text{не зависит от } X} + E_{x,y}\Big[\big(E[y|x] - \mu(X)\big)^2\Big]\Big] = \\ &= E_{x,y}\Big[\big(y - E[y|x]\big)^2\Big] + E_{x,y}\Big[E_X\Big[\big(E[y|x] - \mu(X)\big)^2\Big]\Big]. \end{aligned} \tag{5}$$

Преобразуем второе слагаемое:

$$\begin{aligned} &E_{x,y}\Big[E_X\Big[\big(E[y|x] - \mu(X)\big)^2\Big]\Big] = \\ &= E_{x,y}\Big[E_X\Big[\big(E[y|x] - E_X\big[\mu(X)\big] + E_X\big[\mu(X)\big] - \mu(X)\big)^2\Big]\Big] = \\ &= E_{x,y}\Big[E_X\Big[\underbrace{\big(E[y|x] - E_X\big[\mu(X)\big]\big)^2}_{\text{не зависит от } X}\Big]\Big] + E_{x,y}\Big[E_X\Big[\big(E_X\big[\mu(X)\big] - \mu(X)\big)^2\Big]\Big] + \\ &\quad + 2E_{x,y}\Big[E_X\Big[\big(E[y|x] - E_X\big[\mu(X)\big]\big)\big(E_X\big[\mu(X)\big] - \mu(X)\big)\Big]\Big]. \end{aligned} \tag{6}$$

Покажем, что последнее слагаемое обращается в нуль:

$$\begin{aligned} E_X\Big[\big(E[y|x] - E_X\big[\mu(X)\big]\big)\big(E_X\big[\mu(X)\big] - \mu(X)\big)\Big] = \\ = \big(E[y|x] - E_X\big[\mu(X)\big]\big)E_X\Big[E_X\big[\mu(X)\big] - \mu(X)\Big] = \\ = \big(E[y|x] - E_X\big[\mu(X)\big]\big)\Big[E_X\big[\mu(X)\big] - E_X\big[\mu(X)\big]\Big] = \\ = 0. \end{aligned}$$

Учитывая это, подставим (6) в (5):

$$\begin{aligned} L(\mu) = \underbrace{E_{x,y}\Big[\big(y - E[y|x]\big)^2\Big]}_{\text{шум}} + \\ + \underbrace{E_x\Big[\big(E_X\big[\mu(X)\big] - E[y|x]\big)^2\Big]}_{\text{смещение}} + \underbrace{E_x\Big[E_X\Big[\big(\mu(X) - E_X\big[\mu(X)\big]\big)^2\Big]\Big]}_{\text{разброс}}. \end{aligned} \tag{7}$$

Логика:

- Шум показывает ошибку **лучшей модели** $= E(y|x)$, это шум исходных данных, устранить не выйдет
- Смещение показывает, насколько наша модель (среднее по обученным алгоритмам $\mu(X^n)$) отличается от лучшего алгоритма $E(y|x)$
- Дисперсия показывает разброс ответов обученных алгоритмов относительно среднего ответа

Разложение верно почти для любой функции потерь (у нас была квадратичная).

13.3 Разложение для бэггинга

В *бэггинге (bagging, bootstrap aggregation)* предлагается обучить некоторое число алгоритмов $b_n(x)$ с помощью метода $\tilde{\mu}$, и построить итоговую композицию как среднее данных базовых алгоритмов:

$$a_N(x) = \frac{1}{N}\sum_{n=1}^{N} b_n(x) = \frac{1}{N}\sum_{n=1}^{N} \tilde{\mu}(X)(x).$$

Заметим, что в методе обучения для бэггинга появляется ещё один источник случайности — взятие подвыборки. Чтобы функционал качества $L(\mu)$ был детерминированным, мы будем далее считать, что матожидание $E_X[\cdot]$ берётся не только по всем обучающим выборкам X, но ещё и по всем возможным подвыборкам $\tilde{X}$, получаемым с помощью бутстрапа. Это вполне логичное обобщение, поскольку данное матожидание вводится в функционал именно для учёта случайностей, связанных с процедурой обучения модели.

Найдём смещение из разложения (7) для бэггинга:

$$E_{x,y}\Big[\Big(E_X\Big[\frac{1}{N}\sum_{n=1}^{N}\tilde{\mu}(X)(x)\Big] - E[y|x]\Big)^2\Big] =$$
$$= E_{x,y}\Big[\Big(\frac{1}{N}\sum_{n=1}^{N}E_X[\tilde{\mu}(X)(x)] - E[y|x]\Big)^2\Big] =$$
$$= E_{x,y}\Big[\big(E_X\big[\tilde{\mu}(X)(x)\big] - E[y|x]\big)^2\Big].$$

Мы получили, что смещение композиции, полученной с помощью бэггинга, совпадает со смещением одного базового алгоритма. Таким образом, бэггинг не ухудшает смещенность модели.

Теперь перейдём к разбросу. Запишем выражение для дисперсии композиции, обученной с помощью бэггинга:

$$E_{x,y}\Big[E_X\Big[\Big(\frac{1}{N}\sum_{n=1}^{N}\tilde{\mu}(X)(x) - E_X\Big[\frac{1}{N}\sum_{n=1}^{N}\tilde{\mu}(X)(x)\Big]\Big)^2\Big]\Big].$$

Рассмотрим выражение, стоящее под матожиданиями:

$$\Big(\frac{1}{N}\sum_{n=1}^{N}\tilde{\mu}(X)(x) - E_X\Big[\frac{1}{N}\sum_{n=1}^{N}\tilde{\mu}(X)(x)\Big]\Big)^2 =$$
$$= \frac{1}{N^2}\Big(\sum_{n=1}^{N}\Big[\tilde{\mu}(X)(x) - E_X\big[\tilde{\mu}(X)(x)\big]\Big]\Big)^2 =$$
$$= \frac{1}{N^2}\sum_{n=1}^{N}\Big(\tilde{\mu}(X)(x) - E_X\big[\tilde{\mu}(X)(x)\big]\Big)^2 +$$
$$+ \frac{1}{N^2}\sum_{n_1\neq n_2}\Big(\tilde{\mu}(X)(x) - E_X\big[\tilde{\mu}(X)(x)\big]\Big)\Big(\tilde{\mu}(X)(x) - E_X\big[\tilde{\mu}(X)(x)\big]\Big)$$

Возьмем теперь матожидания от этого выражения, учитывая, что все базовые алго-

ритмы одинаково распределены относительно X:

$$E_{x,y}\Big[E_X\Big[\frac{1}{N^2}\sum_{n=1}^{N}\Big(\tilde{\mu}(X)(x)-E_X\big[\tilde{\mu}(X)(x)\big]\Big)^2+$$
$$+\frac{1}{N^2}\sum_{n_1\neq n_2}\Big(\tilde{\mu}(X)(x)-E_X\big[\tilde{\mu}(X)(x)\big]\Big)\Big(\tilde{\mu}(X)(x)-E_X\big[\tilde{\mu}(X)(x)\big]\Big)\Big]\Big]=$$
$$=\frac{1}{N^2}E_{x,y}\Big[E_X\Big[\sum_{n=1}^{N}\Big(\tilde{\mu}(X)(x)-E_X\big[\tilde{\mu}(X)(x)\big]\Big)^2\Big]\Big]+$$
$$+\frac{1}{N^2}E_{x,y}\Big[E_X\Big[\sum_{n_1\neq n_2}\Big(\tilde{\mu}(X)(x)-E_X\big[\tilde{\mu}(X)(x)\big]\Big)\times$$
$$\times\Big(\tilde{\mu}(X)(x)-E_X\big[\tilde{\mu}(X)(x)\big]\Big)\Big]\Big]=$$
$$=\frac{1}{N}E_{x,y}\Big[E_X\Big[\Big(\tilde{\mu}(X)(x)-E_X\big[\tilde{\mu}(X)(x)\big]\Big)^2\Big]\Big]+$$
$$+\frac{N(N-1)}{N^2}E_{x,y}\Big[E_X\Big[\Big(\tilde{\mu}(X)(x)-E_X\big[\tilde{\mu}(X)(x)\big]\Big)\times$$
$$\times\Big(\tilde{\mu}(X)(x)-E_X\big[\tilde{\mu}(X)(x)\big]\Big)\Big]\Big]$$

Первое слагаемое — это дисперсия одного базового алгоритма, деленная на длину композиции N. Второе — ковариация между двумя базовыми алгоритмами. Мы видим, что если базовые алгоритмы некоррелированы, то дисперсия композиции в N раз меньше дисперсии отдельных алгоритмов. Если же корреляция имеет место, то уменьшение дисперсии может быть гораздо менее существенным.

14 Построение ансамблей. Бэггинг и случайный лес, Градиентный бустинг, Stacking.

14.1 Построение ансамблей

Ансамблем (Ensemble, Multiple Classifier System) называется алгоритм, который состоит из нескольких алгоритмов машинного обучения, а процесс построения ансамбля называется ансамблированием (ensemble learning).

Простейший пример ансамбля в регрессии – усреднение нескольких алгоритмов:

$$a(x) = 1/n * (b_1(x) + b_2(x) + ... + b_k(x))$$

Для понимания: Для построения ансамбля достаточно в каком-то количестве взять какие-то алгоритмы и как-то соединить их ответы: случайный лес - это ансамбль деревьев. Усреднённые Knn также будет ансамблем. Бустинг - это тоже типичный пример ансамлбля. Нейронная сеть - это тоже ансамбль (однородный).

14.2 Бэггинг и случайный лес

Бэггинг (**B**ootstrap **Agg**regat**ing**) – способ построения ансамбля следующим образом: мы строем базовую модель на бутстреп подвыборке, делаем это несколько на разных

подвыборках, получаем разные модели, после чего применяем к данным и усредняем ответы

При отборе подвыборки с помощью бутстрепа вероятность какого-либо элемента из изначального множества попасть в подвыборку составляет при каком-либо взятии $\frac{1}{n}$, тогда вероятность не попасть выборку при этом взятии $1 - \frac{1}{n}$. Отсюда получаем, что вероятность не попасть в бутстреп подвыборку того же размера, что и изначальная составляет $\left(1 - \frac{1}{n}\right)^n$, значит, вероятность попасть хотя бы раз будет:

$$1 - \left(1 - \frac{1}{n}\right)^n \longrightarrow_{n\to\infty} 1 - \frac{1}{e} \approx 0{,}63$$

Получается, что каждая подвыборка состоит примерно из 63% оригинальной выборки, на которой и обучается, остальная часть называется out-of-bag-наблюдениями. На этих наблюдениях можно оценивать качество моделей, считая для них ответы, а с их помощью и ошибки. Ответ для таких наблюдений получаем следующим образом:

$$a_{OOB}(x_j) = \frac{1}{|\{i : x_j \in OOB_i\}|} \sum_{i:x_j \in OOB_i} b_i(x_j)$$

Случайный лес
Алгоритм:

1) Берём бутстрап подвыборку

2) Строим на подвыборке случайное дерево следующим образом: сначала выбираем случайное подмножество фичей, потом по ним ищем оптимальный сплит, после чего опять выбираем случайное подмножество фичей и уже по ним делаем оптимальный сплит и так далее

3) Повторяем шаги 1), 2) для постройки необходимого нам количества деревьев

4) Получаем ответы случайного леса, как среднее предсказание по всем деревьям

14.3 Градиентный бустинг

Бустинг — метод ансамблирования, в котором мы хотим получить алгоритм следующего вида:

$$a_N(x) = \sum_{j=0}^{N} b_j(x)$$

$b_j(x)$ - это простые алгоритмы (weak learners), которые обучаются итеративно (то есть сначала мы назначаем нулевой алгоритм $b_0(x)$, а потом последовательно обучаем первый, второй и т.д.) по следующему правилу:

$$\left[\begin{array}{lcl} s_i^{(N)} & = & y_i - a_N(x_i) \\ b_N(x) & = & \underset{b\in\mathcal{A}}{\operatorname{argmin}} \sum_{i=1}^{n} (b(x_i) - s_i^{(N)})^2 \end{array}\right.$$

Теперь для градиентного бустинга немного изменим модель добавив в неё веса γ_j:

$$a_N(x) = \sum_{j=0}^{N} \gamma_j b_j(x)$$

Применим нашу loss-функцию на наш алгоритм и воспользуемся свойством того, что мы составные алгоритмы обучаем по очереди:

$$\sum_{i=1}^{n} \mathcal{L}y_i,a_N(x_i)) \to \min$$

$$\sum_{i=1}^{n} \mathcal{L}y_i,a_{N-1}(x_i) + \gamma_N b_N(x_i)) \to \min_{\gamma_N b_N}$$

Если loss-функция квадратичная ($\mathcal{L}(y,z) = \frac{1}{2}(y-z)^2$), то тогда получим, что $s_i^{(N)}$ является антиградиентом этой loss-функции в точки предсказания нашей модели:

$$s_i^{(N)} = -\frac{\partial}{\partial z}\mathcal{L}(y,z)\Big|_{z=a_{N-1}(x_i)}$$

Сделаем замену $\gamma_N b_N(x_i) = s_i$. Тогда получим, что максимизация будет устроена следующим образом:

$$\sum_{i=1}^{n} \mathcal{L}y_i,a_{N-1}(x_i) + s_i) \to \min_{s_1,\ldots,s_n}$$

$$s_i = y_i - a_{N-1}(x_i)$$

Однако такой подход будет плох, так как он не учитывает метрику (работает хорошо с квадратичной), и будет возникать переобучение. Тогда наилучшим выбором будет следовать по антиградиенту loss-функции.

Тогда применим формулу Тейлора для случая многомерной функции и поставим оптимизационную задачу:

$$f(x+s) = f(x) + Df(x)^T s + o(||s||)$$

$$\begin{cases} f(x+s) \to \min\limits_{s} \\ ||s|| = 1 \end{cases}$$

Из минимизации получаем:

$$s = -\frac{Df(x)}{||Df(x)||}$$

Алгоритм градиентного бустинга:

1) Вычисляем антиградиенты нашей функции потерь в точках ответов нашей предыдущей компазиции (которая уже настроена)

$$s_i^{(N)} = -\frac{\partial}{\partial z}\mathcal{L}(y,z)\Big|_{z=a_{N-1}(x_i)}$$

2) Настраиваем наши слабые модели (weak learners)

$$b_N(x) = \underset{b\in\mathcal{A}}{\operatorname{argmin}} \sum_{i=1}^{n} (b(x_i) - s_i^{(N)})^2$$

3) Настраиваем веса (гамму)

$$\gamma_N(x) = \underset{\gamma\in R}{\operatorname{argmin}} \sum_{i=1}^{n} \mathcal{L}(y_i, a_{N-1}(x_i) + \gamma b_N(x_i))$$

Особенности градиентного бустинга над деревьями:

Когда мы используем деревья, то мы можем представить простую модель (дерево) в следующем виде:

$$b(x) = \sum_{t=1}^{T} w_t [x \in R_t]$$

Тогда наша функция потерь выглядит следующим образом:

$$\sum_{i=1}^{n} \mathcal{L}(y_i, a_{N-1}(x) + \gamma_N \sum_{t=1}^{T} w_{Nt}[x \in R_t])$$

Однако и γ_N и w_{Nt} представляют из себя константы, а потому нет смысла сначала искать одну, потом другую, проще их сразу объединить, после чего мы можем разбить нашу задачу на эквивалентные подзадачи, так как константа ищется для каждого листа отдельно, независимо от других:

$$\sum_{i:(x_i,y_i)\in R_t} \mathcal{L}(y_i, a_{N-1}(x_i) + w_{Nt})$$

Тогда алгоритм получается следующий:

1) Вычисляем антиградиенты нашей функции потерь в точках ответов нашей предыдущей компазиции (тут ничего не поменялось, так как антиградиенты зависят от функции потерь, а не от моделей)

$$s_i^{(N)} = -\frac{\partial}{\partial z}\mathcal{L}(y,z)\Big|_{z=a_{N-1}(x_i)}$$

2) Настраиваем наши деревья (которые соответсвуют weak learners, и которые обучаются по антиградиентам)

$$b_N(x) = \underset{b\in DecisionTree}{\operatorname{argmin}} \sum_{i=1}^{n} (b(x_i) - s_i^{(N)})^2$$

3) Обновляем константы в листьях в соответствии с нашей функцией потерь (то есть деревья у нас обучаются и выдают константу оптимальную для MSE, а мы их меняем а не оптимальные с точки зрения функции потерь)

$$w_{Nt}^*(x) = \underset{w\in R}{\operatorname{argmin}} \sum_{i:(x_i,y_i)\in R_t} \mathcal{L}(y_i, a_{N-1}(x_i) + w)$$

Пример для понимания: из второго шага мы получили константы, и так как у нас во втором шаге минимизируется квадрат разницы, то константа равна среднему значению, теперь представим, что в качестве функции потерь у нас взят модуль отклонения, тогда получим, что оптимальная константа для функции потерь должна быть равна медианному значению

14.4 Stacking

Стекинг (Stacked Generalization или Stacking) — один из самых популярных способов ансамблирования алгоритмов, т.е. использования нескольких алгоритмов для решения одной задачи машинного обучения. Основная идея стекинга - это построение метапризнаков, то есть ответов базовых алгоритмов на объектах выборки, и обучение на них мета- алгоритма. В общем виде формально записывается в следующем виде:

Это функция от ответов других алгоритмов

$$a(x) = d(b_1(x), b_2(x),...,b_k(x))$$

Алгоритм принимает на вход ответы других алгоритмов, как признаки

Графическая иллюстрация стэкинга

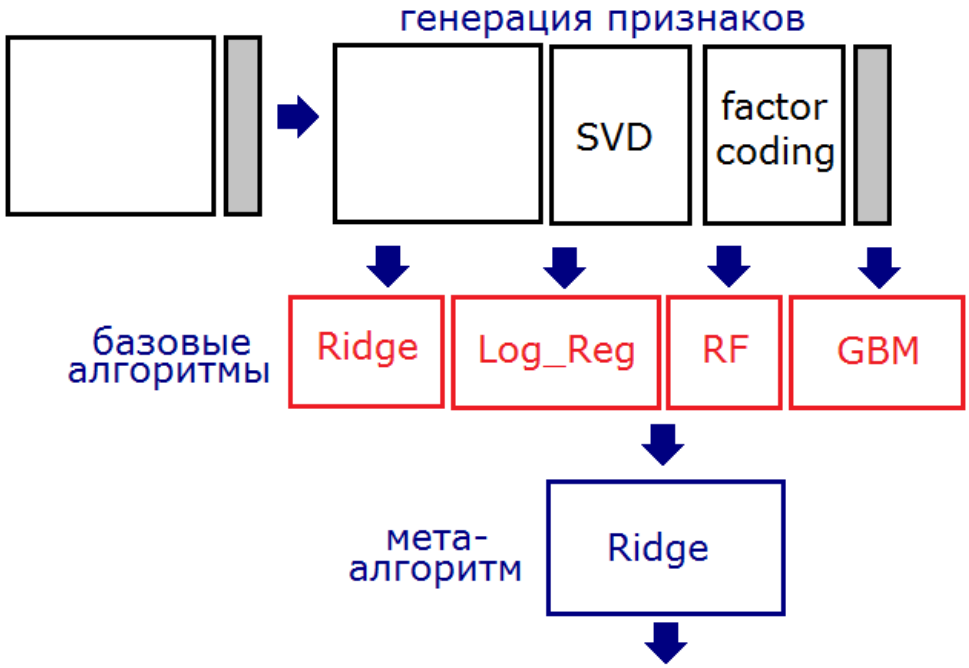


Рис. 8. Стандартное изображение стекинга

Простейшая схема стекинга — блендинг (Blending): обучающую выборку делят на две части. На первой обучают базовые алгоритмы. Затем получают их ответы на второй части и на тестовой выборке. Понятно, что ответ каждого алгоритма можно рассматривать как новый признак (т.н. «метапризнак»). На метапризнаках второй части обучения настраивают метаалгоритм. Затем запускают его на метапризнаках

теста и получают ответ. В рамках блендинга ответы алгоритмов просто усредняются, тогда как в стекинге они используются как входные данные для нового алгоритма.

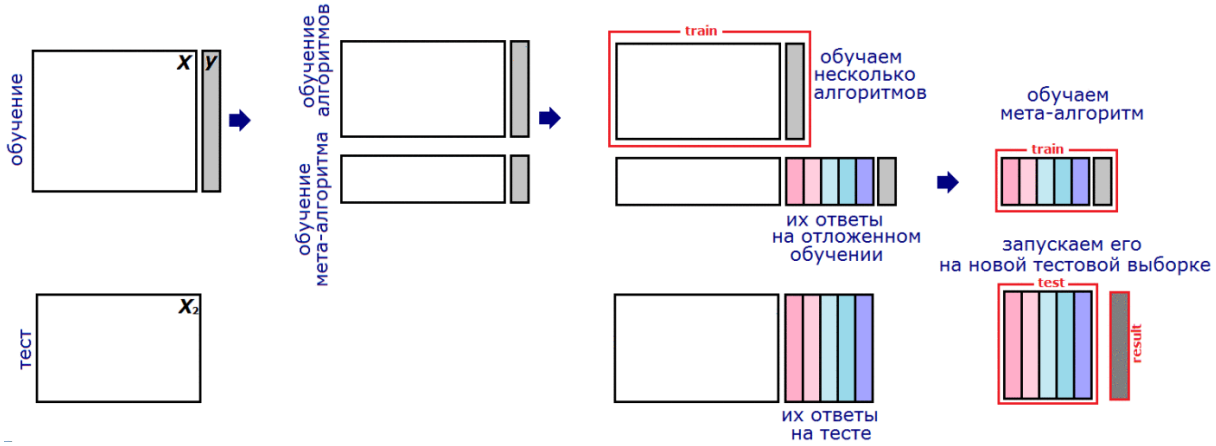


Рис. 9. Схема классического блендинга

Заметна проблема нерационального использования выборки, поэтому для повышения качества надо усреднить несколько блендингов с разными разбиениями обучения. Вместо усреднения иногда конкатенируют обучающие (и тестовые) таблицы для метаалгоритма, полученные при разных разбиениях: здесь мы получаем несколько ответов для каждого объекта тестовой выборки — их усредняют. На практике такая схема блендинга сложнее в реализации и более медленная, а по качеству может не превосходить обычного усреднения.

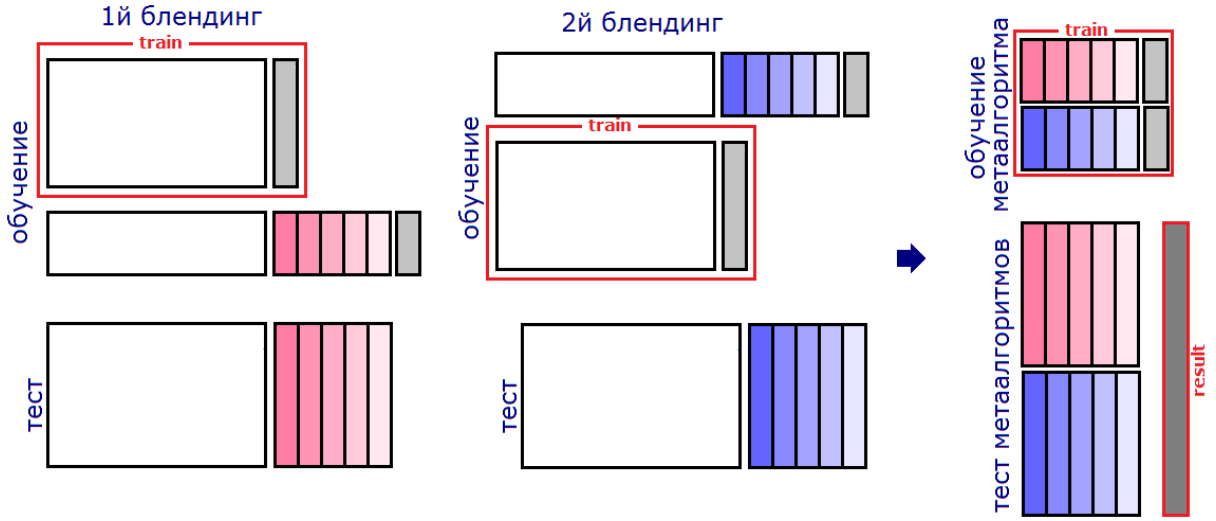


Рис. 10. Возможная модификация блендинга

В рамках классического стэкинга выборку разбивают на части (фолды), затем последовательно перебирая фолды обучают базовые алгоритмы на всех фолдах, кроме одного, а на оставшемся получают ответы базовых алгоритмов и трактуют их как значения соответствующих признаков на этом фолде. Для получения метапризнаков объектов тестовой выборки базовые алгоритмы обучают на всей обучающей выборке

и берут их ответы на тестовой. Самый главный недостаток (классического) стекинга в том, что метапризнаки на обучении (пусть и полноценном — не урезанном) и на тесте разные.Часто с указанным недостатком борются обычной регуляризацией.

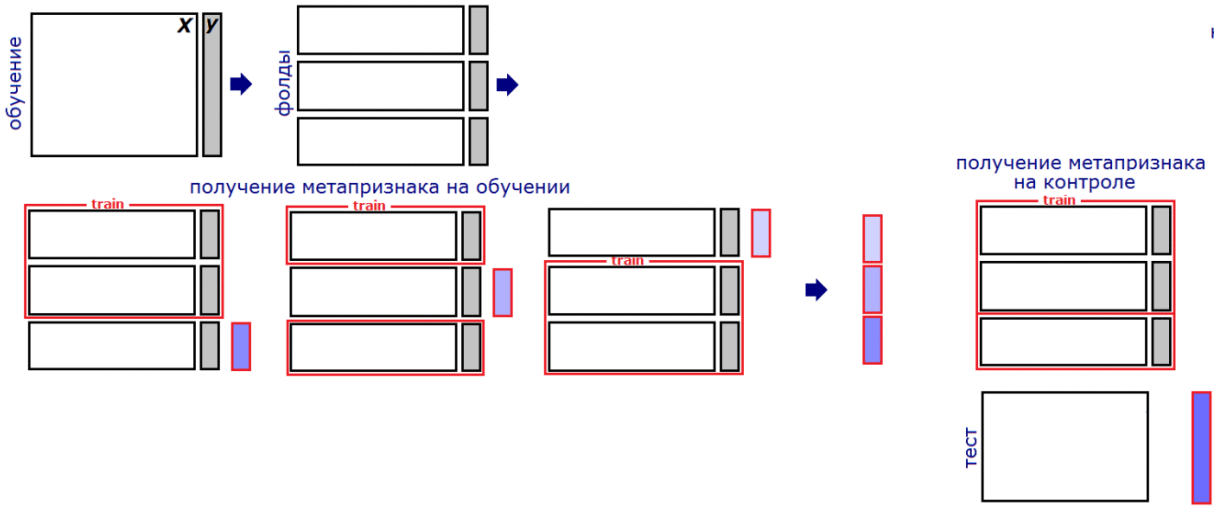


Рис. 11. Получение метапризнака в классическом стекинге

15 Полносвязные нейронные сети. Функции активации, функции потерь. Автоматическое дифференцирование. Прямой проход по сети. Обратный проход по сети. Решение различных задач машинного обучения с помощью нейронных сетей: снижение размерности, классификация. Архитектура Автокодировщика.

15.1 Полносвязные нейросети

Полносвязная нейросеть (fully connected neural network) – класс искусственных нейронных сетей прямого распространения (ациклический граф), состоящих как минимум из трех слоёв: входного (input layer), скрытого (hidden layer) и выходного (output layer) (рис. 12). Специфика данных нейросетей состоит в том, что каждый нейрон одного слоя имеет связь с каждым нейроном следующего за ним слоя. Такие слои в свою очередь называют полносвязными.

Каждый слой имеет какое-то количество нейронов (кружочки на рисунке). Количество нейронов на входном слое равно количеству располагаемых признаков (размерности признакового пространства), количество нейронов на выходном слое зависит от задачи: в случаи обычной **регрессии и бинарной классификации** нейрон будет один, в случае **многоклассовой классификации** количество нейронов соответствует количеству классов (для бинарной тоже будет верно). Нейроны на скрытых и выходном слоях аккумулируют информацию с предыдущего им слоя (суммируют-

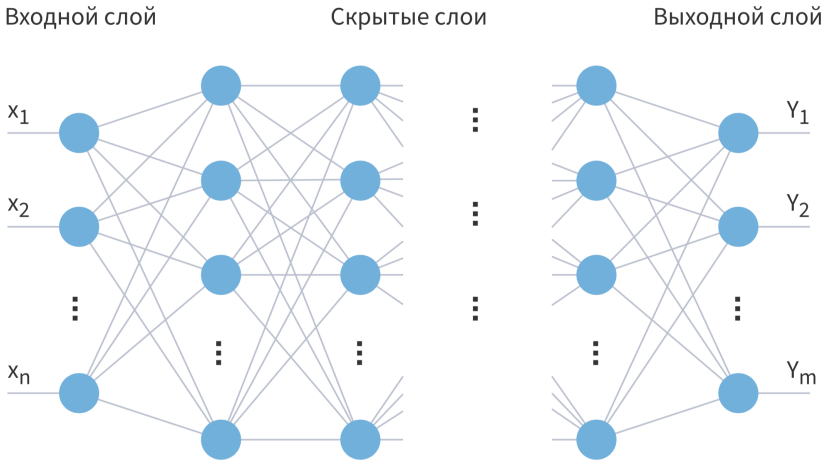


Рис. 12. Красота какая

ся в виде линейной комбинации со сдвигом) и применяют **функции активации**, например:

- $\sigma(z) = \frac{1}{1+e^{-z}}$ – сигмоида;
- $\text{th}(z) = \frac{e^z - e^{-z}}{e^z + e^{-z}}$ – гиперболический тангенс;
- $f(z) = \max(0, z)$ – Rectified linear unit, ReLU

В качестве активации на выходном слое в случае бинарной классификации используется сигмоидная функция, в случае регрессии годится и просто тождественное преобразование ($f(z) = z$), а в случае многоклассовой используется Softmax преобразование:

$$\sigma(\mathbf{z})_i = \frac{e^{z_i}}{\sum_{k=1}^{K} e^{z_k}}$$

На словах, в каждом нейроне берётся экспоненциальная функция от z_j (z_j уже является взвешенной суммой значений нейронов с прошлого слоя со сдвигом, то есть $z_j = \sum_{i=1}^{d} w_i \cdot x_i + b$, где d – количество нейронов в предыдущем слое) и нормируется.

В качестве **функций потерь** обычно используют:

- в случае регрессии пойдёт классический MSE;
- в случае многоклассовой классификации (бинарная будет частным случаем) используется кросс-энтропия:

$$logloss = -\frac{1}{n} \sum_{i=1}^{n} \sum_{j=1}^{l} y_{ij} \cdot \log a_{ij}, \text{ где}$$

n – число наблюдений, l – число классов, y_{ij} равен 1, если i-ое наблюдение принадлежит классу j, и 0 иначе, a_{ij} – модельная вероятность принадлежности i-ое наблюдения к j-ому классу.

15.2 Обучение нейросети

Автоматическое дифференцирование – метод обучения нейросетей, основанный на двух шагах:

1. **Прямое распространение** (forward propagation) – процесс пропуска входных данных через нейросеть, попутно сохраняя промежуточные ответы и значение функции ошибки;

2. **Обратное распространение** (backward propagation) – процесс подсчёта градиентов с помощью chain rule.

Благодаря автоматическому дифференцированию можно обучать нейросети любой сложности, так как функции будут раскладываться в элементарные с помощью chain rule. Более того, мы не будем делать подсчёт лишних операций, так как градиенты с верхних слоёв будут сохраняться и утилизироваться далее.

15.3 Снижение размерности, автокодировщик

Автоэнкодер – специальный вид нейросетей, где количество нейронов в выходном слое соответствует количеству нейронов в входном: нейросеть пытается копировать входные данные на выход как можно точнее (рис. 13). Encoder часть автоэнкодера сжимает входные данные для представления их в latent-space (скрытое пространство), а затем decoder часть восстанавливает из этого представления output (выходные данные). Частный случай полносвязной нейросети вполне себе пример такого автокодера.

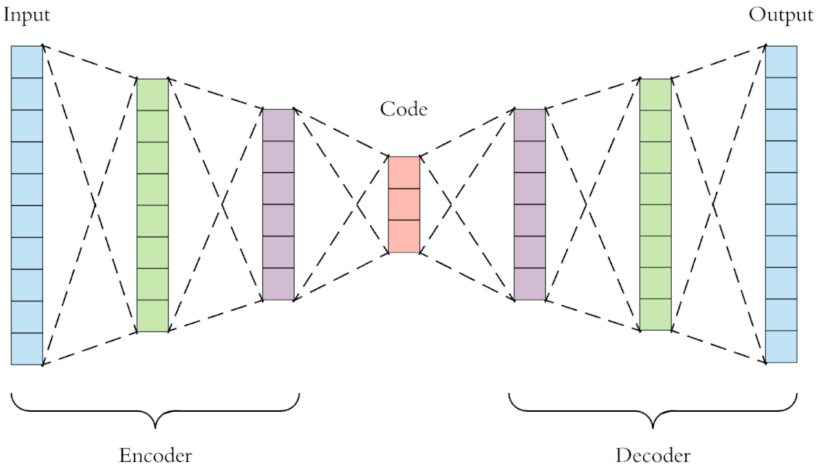


Рис. 13. Ешё одна красота

Данный вид нейросетей используется для некоторых задач обучения без учителя: **уменьшения шума в данных и снижения размерности** многомерных данных для визуализации (оказывается лучше классических подходов типа PCA и других). Для целей визуализации используется скрытое представление (оранжевые нейроны на картинке), которое пытается сохранить в себе как можно больше информации (так как дальше на её основе происходило восстановление).

16 Список литературы

[1] Weber, R., Schek, H. J., Blott, S. (1998). A Quantitative Analysis and Performance Study for Similarity-Search Methods in High-Dimensional Spaces. // Proceedings of the 24th VLDB Conference, New York C, 194–205.