Оглавление

Машинное обучение - 2 ФКН ВШЭ ПМИ

Коллоквиум Весна

vladmalkov, and others

0.1 Определение ядра через скалярное произведение и теорему Мерсера. Определения полиномиального и гауссова ядра. Каким спрямляющим пространствам они соответствуют?

Ядра

Ядром мы будем называть функцию $K(x,z)$, представимую в виде скалярного произведения в некотором пространстве: $K(x,z) = \langle \phi(x), \phi(z) \rangle$, где $\phi : X \to H$ — отображение из исходного признакового пространства в некоторое *спрямляющее пространство*. На семинарах будет показано, что ядро содержит в себе много информации о спрямляющем пространстве и позволяет производить в нем различные операции, не зная самого отображения $\phi(x)$ — например, находить расстояния между векторами $\phi(x)$.

Самый простой способ задать ядро — в явном виде построить отображение $\phi(x)$ в спрямляющее признаковое пространство. Тогда ядро определяется как скалярное произведение в этом пространстве: $K(x,z) = \langle \phi(x), \phi(z) \rangle$. При таком способе, однако, возникают проблемы с ростом вычислительной сложности, о которых уже было сказано выше. Пользоваться этим на практике сложно, поэтому для построения ядер используются некоторые базовые вещи:

Пусть нам даны ядра $K_1(x,z), K_2(x,z)$ на пространстве X, $f : X \to \mathbb{R}$ - вещественная функция на X, $\phi : X \to \mathbb{R}^N$ - векторная функция на X и K_3 - ядро на пространстве $\mathbb{R}^N$. Тогда ядро можно получить с помощью следующих операций:

1. $K(x,z) = K_1(x,z) + K_2(x,z)$
2. $K(x,z) = \alpha K_1(x,z), \alpha > 0$
3. $K(x,z) = K_1(x,z) \cdot K_2(x,z)$
4. $K(x,z) = f(x) \cdot f(z)$
5. $K(x,z) = K_3(\phi(x), \phi(z))$

Пусть $K_1(x,z), K_2(x,z)...$ - последовательность ядер, причем $K(x,z) = \lim_{n\to\infty} K_n(x,z)$ $\exists \forall x, z$. Тогда $K(x,z)$ - ядро.

Неявное задание ядра

Как убедиться, что функция $K(x,z)$ определяет скалярное произведение в некотором пространстве? Ответ на этот вопрос дает *теорема Мерсера*: функция $K(x,z)$ является ядром тогда и только тогда, когда:

1. Она симметрична: $K(x,z) = K(z,x)$.
2. Она неотрицательно определена, то есть для любой конечной выборки $(x_1, \ldots, x_\ell)$ матрица $K = \big(K(x_i, x_j)\big)_{i,j=1}^{\ell}$ неотрицательно определена.

Полиномиальное ядро

Пусть $p(v)$ - многочлен с положительными коэффициентами. Очевидно, $p(K(x,z))$ является ядром для любого $K(x,z)$.

Полиномиальное ядро степени m можно определить как $K_m(x,z) = (\langle x, z \rangle + R)^m$. Согласно формуле бинома,

$$K_m(x,z) = \sum_{i=0}^{m} C_m^i R^{m-i} \langle x, z \rangle^i$$

Все коэффициенты положительны, поэтому это действительно ядро. Оно соответствует переводу признаков во все возможные наборы мономов над признаками степени не более m.

Гауссово ядро определяется как $K(x,z) = \exp\left(-\frac{||x-z||^2}{2\sigma^2}\right)$. Докажем, что это действительно ядро:

$$\exp\left(\langle x,z\rangle\right) = \sum_{k=0}^{\infty} \frac{\langle x,z\rangle^k}{k!} = \lim_{n\to\infty} \sum_{k=0}^{n} \frac{\langle x,z\rangle^k}{k!}$$

Каждый член этой числовой последовательности - ядро как многочлен с положительными коэффициентами, предел существует во всех точках (т.к. ряд Тейлора экспоненты сходится всюду) - отсюда получаем, что экспонента скалярного произведения является ядром.

Аналогично доказывается ядерность $\exp\left(\frac{\langle x,z\rangle}{\sigma^2}\right)$.

Осталось перейти к исходному ядру, для этого сделаем так

$$||x-z||^2 = \langle x,x\rangle + \langle z,z\rangle - 2\langle x,z\rangle$$

$$\exp\left(-||x-z||^2\right) = \frac{\exp(2\langle x,z\rangle)}{\exp(||x||^2)\exp(||z||^2)}$$

Сверху ядро, снизу произведение вещественных функций - ядро, общее произведение - ядро.

Это ядро задает бесконечномерное полиномиальное пространство (доказывается Тейлором).

0.2 Что такое двойственная задача оптимизации. Условия оптимальности Куна-Таккера — для чего они нужны? (знать наизусть условия не требуется, достаточно рассказать, что именно они дают и как могут быть использованы для поиска оптимума)

2. Задача условной оптимизации. Двойственная задача. Теорема Куна-Таккера.

Запишем некоторую задачу оптимизации с ограничениями:

$$\begin{cases} f_0(x) \to \min\limits_{x\in\mathbb{R}^d} \\ f_i(x) \le 0, \qquad \forall i \in \{1,...,m\} \\ h_i(x) = 0, \qquad \forall i \in \{1,...,p\} \end{cases} \tag{1}$$

Собственно, это и есть задача условной оптимизации.

Из нее можно получить задачу безусловной оптимизации:

$$f_0(x) + \sum_{i=1}^{m} I_-(f_i(x)) + \sum_{i=1}^{p} I_0(h_i(x)) \to \min_x$$

$$I_-(x) = \begin{cases} 0, & x \le 0 \\ \infty, & x > 0 \end{cases}$$

$$I_0(x) = \begin{cases} 0, & x = 0 \\ \infty, & x \ne 0 \end{cases}$$

Но это сложно оптимизировать, поэтому заменим индикаторы на линейные аппроксимации:

$$L(x, \lambda, \mu) = f_0(x) + \sum_{i=1}^{m} \lambda_i f_i(x) + \sum_{i=1}^{p} \mu_i h_i(x)$$

Выражение выше называется лагранжианом исходной задачи, а новые коэффициенты λ, μ - двойственными переменными.

Теперь мы можем выразить через него двойственную функцию исходной задачи:

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

Эта функция дает нижнюю оценку на минимум в исходной задаче.

$$g(\lambda, \mu) \leq f_0(x_*)$$

[Входит в теормин]

Соответственно, двойственную к исходной задачу можно представить как максимизацию двойственной функции с ограничениями на новые коэффициенты:

$$\begin{cases} g(\lambda, \mu) \to \max\limits_{\lambda, \mu} \\ \lambda_i \geq 0, \qquad\qquad i \in \{1, ..., m\} \end{cases} \tag{2}$$

[]

Очевидно, для любого решения двойственной задачи (λ^*, μ^*) значение двойственной функции не превосходит условный минимум исходной задачи:

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

Это свойство называется слабой двойственностью, в случае равенства имеет место сильная двойственность. Разность между правой и левой частью называется двойственным зазором.

Для сильной двойственности в выпуклых задачах имеются различные достаточные условия. Сформулируем одно из них.

Задача оптимизации называется выпуклой, если она имеет следующий вид:

$$\begin{cases} f_0(x) \to \min\limits_{x \in \mathbb{R}^d} \\ f_i(x) \leq 0, \qquad i \in \{1, ..., m\} \\ Ax = b \end{cases} \tag{3}$$

где функции $f_0, f_1, ..., f_m$ являются выпуклыми. Условие Слейтера требует существование такой точки x', в которой все ограничения бы выполнялись, при этом ограничения-неравенства были бы выполнены строго. Впрочем, достаточно, чтобы эти неравенства выполнялись строго, только если они имеют нелинейный вид.

Пусть у нас есть решения прямой и двойственной задач x_* и (λ^*, μ^*) соответственно, и пусть имеет место сильная двойственность. В таком случае

$$f_0(x_*) = g(\lambda^*, \mu^*) = \inf_x \left(f_0(x) + \sum_{i=1}^{m} \lambda_i^* f_i(x) + \sum_{i=1}^{p} \mu_i^* h_i(x) \right) \leq$$

$$\leq f_0(x_*) + \sum_{i=1}^{m} \lambda_i^* f_i(x_*) + \sum_{i=1}^{p} \mu_i^* h_i(x_*) \leq f_0(x_*)$$

Все неравенства здесь превращаются в равенства. Отсюда мы можем получить, что $\lambda_i^* f_i(x_*) = 0 \ \forall i \in \{1, ..., m\}$ (следует из равенства суммы нулю и неположительности каждого слагаемого). Эти условия называются условиями

дополняющей нежесткости, согласно им множитель λ при $i-$ом ограничении может быть ненулевым, только если оно выполняется с равенством (является активным).

[Входит в теормин]

Запишем условия, выполненные для решения прямой и двойственной задач:

$$\begin{cases} \nabla f_0(x_*) + \sum\limits_{i=1}^{m} \lambda_i^* \nabla f_i(x_*) + \sum\limits_{i=1}^{p} \mu_i^* \nabla h_i(x_*) = 0 & \\ f_i(x_*) \le 0, & i \in \{1, ..., m\} \\ h_i(x_*) = 0, & i \in \{1, ..., p\} \\ \lambda_i^* \ge 0, & i \in \{1, ..., m\} \\ \lambda_i^* f_i(x_*) = 0, & i \in \{1, ..., m\} \end{cases} \tag{4}$$

Эти условия называются условиями Каруша-Куна-Таккера и являются необходимыми условиями экстремума. Можно сформулировать в виде теоремы: пусть есть решение x_*, тогда найдутся такие λ^*, μ^*, что выполнены условия ККТ. Если задача является выпуклой и удовлетворяет условиям Слейтера, то условия ККТ будут необходимыми и достаточными.

0.3 EM-алгоритм: шаги, оптимизируемый функционал.

EM-алгоритм — итерационный метод максимизации правдоподобия выборки. Пусть есть следующая задача:

$$\log p(X|\Theta) \to \max_{\Theta} \tag{5}$$

Пусть в модели существуют скрытые переменные Z, описывающее её внутреннее состояние. Для некоторого распределения $q(Z)$ на скрытых переменных верно:

$$\begin{aligned} \log p(X|\Theta) = \int q(Z) \log p(X|\Theta) dZ = \\ \int q(Z) \log \frac{p(X,Z|\Theta)}{p(Z|X,\Theta)} dZ = \int q(Z) \log \frac{p(X,Z|\Theta) q(Z)}{p(Z|X,\Theta) q(Z)} dZ = \\ \int q(Z) \log \frac{p(X,Z|\Theta)}{q(Z)} dZ + \int q(Z) \log \frac{q(Z)}{p(Z|X,\Theta)} dZ = \\ \mathcal{L}(q,\Theta) + \mathrm{KL}(q\|p). \end{aligned}$$

Так как $\mathrm{KL}(q\|p) \ge 0$, то $\log p(X|\Theta) \ge \mathcal{L}(q,\Theta)$.

Напомним, что мы хотели бы максимизировать левую часть получившегося неравенства, не зависящую от распределения q, которое, в свою очередь, может быть выбрано произвольно, поэтому чем «правильнее» будет выбрано q, тем точнее будет полученная нижняя оценка в правой части неравенства. Вместо решения исходной задачи 5 будем максимизировать нижнюю оценку $\mathcal{L}(q,\Theta)$ поочерёдно по q и по Θ.

E-step. Максимизируем по q.

Из полученного ранее следует, что максимум $\mathcal{L}(q, \Theta)$ по q достигается в том случае, когда достигается минимум $\mathrm{KL}(q\|p)$, то есть когда $q = p$:

$$q^*(Z) = \underset{q}{\operatorname{argmax}} \mathcal{L}(q, \Theta^{\text{old}}) = \underset{q}{\operatorname{argmin}} \int q(Z) \log \frac{q(Z)}{p(Z|X, \Theta^{\text{old}})} dZ = p(Z|X, \Theta^{\text{old}})$$

M-step. Максимизируем по Θ.

$$\begin{aligned}\Theta^{\text{new}} &= \underset{\Theta}{\operatorname{argmax}} \int q^*(Z) \log \frac{p(X, Z|\Theta)}{q^*(Z)} dZ = \underset{\Theta}{\operatorname{argmax}} \int q^*(Z) \log p(X, Z|\Theta) dZ \\ &= \underset{\Theta}{\operatorname{argmax}} \mathbb{E}_{Z \sim q^*(Z)} \log p(X, Z|\Theta)\end{aligned}$$

Теперь, сформулируем более четко.

На E-шаге (Expectation) мы выбираем наиболее вероятное распределение скрытых переменных Z - используем байесовский подход и вычислим здесь апостериорное распределение при фиксированных параметрах $p(Z|X, \Theta^{old})$.

На M-шаге (Maximization) мы подбираем такие параметры Θ, которые максимизируют ожидание полного логправдоподобия по апостериорному распределению Z:

0.4 KL-дивергенция: формула, применение для аппроксимации распределений.

KL-дивергенция является мерой расстояния между двумя распределениями.
Формулируется она так:

$$\mathrm{KL}(q||p) = \int_{\mathbb{R}} q(x) \log \frac{q(x)}{p(x)} dx$$

(здесь интеграл берется по всему пространству)

В дискретном случае:

$$\mathrm{KL}(q||p) = \sum_x q(x) \log \frac{q(x)}{p(x)}$$

При вычислении полагаем, что все неопределенности обращаются в 0. Эта мера определена только в том случае, если из $p(x) = 0$ следует $q(x) = 0$.

Применение для аппроксимации распределений:
Чтобы одно распределение приблизить к другому надо минимизировать их KL-дивергенцию Если она равна нулю, то распределения совпадают

0.5 Метод local outlier factor.

Уровень выброса основывается на концепции локальной плотности, где локальность задаётся k ближайшими соседями, расстояния до которых используются для оценки плотности. Путём сравнения локальной плотности объекта с локальной плотностью его соседей можно выделить области с аналогичной плотностью и точки, которые имеют существенно меньшую плотность, чем её соседи. Эти точки считаются выбросами.

Введем определения:

rd - reachibility distance:

$rd_k(x, z) = \max(\rho_k(z), \rho(x, z))$, где $\rho_k(z)$ - расстояние от z до k-го ближайшего соседа z

lrd - local reachibility density:

$\frac{1}{k}\sum_{z\in N_k(x)} rd_k(x,z)$ – среднее расстояние от x до ближайших соседей, то:

$lrd_k(x) = \frac{1}{\frac{1}{k}\sum_{z\in N_k(x)} rd_k(x,z)}$, где $N_k(x)$ - множество k ближайших соседей x

Итого local outlier factor (LOF):

$$LOF_k(x) = \frac{\frac{1}{k}\sum_{z\in N_k(x)} lrd(z)}{lrd(x)}$$

Если $LOF_k(x) > 1 \Rightarrow x-$ выброс

Если $LOF_k(x) < 1 \Rightarrow x-$ внутрениий объект

Значение, примерно равное 1, означает, что объект сравним с его соседями (а тогда он не является выбросом).

0.6 Как устроен одноклассовый SVM?

Для начала вспомним обычный SVM:

$$\begin{cases} \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{\ell}\xi_i \to \min\limits_{w,b,\xi} \\ y_i\left(\langle w, x_i\rangle + b\right) \geq 1 - \xi_i, \quad i = 1,\ldots,\ell, \\ \xi_i \geq 0, \quad i = 1,\ldots,\ell. \end{cases} \tag{6}$$

Здесь $\frac{1}{2}\|w\|^2$ - отвечает за ширину полосы, ξ за допустимые отступы, а C - гиперпараметр, насколько сильно мы хотим штрафовать за отступы. Сильным преимуществом решение задачи SVM был ядровой переход. Если выписать честную двойственную задачу, то оптимизируемые функции будут зависеть только от скалярных произведений объектов выборки, а значит мы можем воспользоваться kernel-trick.

Попробуем обобщить эту идею для случая обнаружения аномалий, только теперь будем пытаться отделять нашу выборку от нуля. Если мы грамотно подберём нашу задачу, то пользуясь kernel-trick отделение от нуля будет выглядеть осмысленным.

Будем строить линейную функцию $a(x) = sign\langle w, x\rangle$, и потребуем, чтобы она отделяла выборку от начала координат с максимальным отступом. Соответствующая оптимизационная задача будет иметь вид:

$$\begin{cases} \frac{1}{2}\|w\|^2 + \frac{1}{\nu\ell}\sum_{i=1}^{\ell}\xi_i - \rho \to \min\limits_{w,\xi,\rho} \\ \langle w, x_i\rangle \geq \rho - \xi_i, \quad i = 1,\ldots,\ell, \\ \xi_i \geq 0, \quad i = 1,\ldots,\ell. \end{cases}$$

Здесь гиперпараметр ν отвечает за корректность на обучающей выборке — можно показать, что он является верхней границей на число аномалий (объектов выборки, на которых $a(x) = -1$). Решающее правило будет иметь вид

$$a(x) = sign\left(\langle w, x\rangle - \rho\right),$$

где ответ -1 будет соответствовать выбросу. Получается, что мы ищем гиперплоскость так, что:

- она отделяет как можно больше объектов выборки от нуля (чем меньше ν, тем больше объектов мы будем отделять) — за это отвечает слагаемое $\frac{1}{\nu\ell}\sum_{i=1}^{\ell}\xi_i$ в функционале;

- она имеет большой отступ $\frac{1}{\|w\|^2}$;
- она при этом как можно сильнее отдалена от нуля (то есть ρ как можно большее значение).

Для данной задачи можно выписать двойственную и сделать ядровой переход в ней:

$$\begin{cases} \frac{1}{2}\sum\limits_{i,j=1}^{\ell}\lambda_i\lambda_j K(x_i,x_j) \to \min\limits_{\lambda} \\ 0 \le \lambda_i \le \frac{1}{\nu\ell}, \quad i = 1,\ldots,\ell, \\ \sum\limits_{i=1}^{\ell}\lambda_i = 1. \end{cases}$$

Модель при этом будет иметь вид

$$a(x) = sign\left(\sum_{i=1}^{\ell}\lambda_i K(x,x_i) - \rho\right).$$

Заметим, что при использовании гауссова ядра данная модель будет очень похожа на метод, который строит непараметрическую оценку плотности (**??**) с гауссовым ядром и сравнивает её значение с порогом ρ.

При использовании подходящих ядер можно действительно получить функцию, которая точно описывает обучающую выборку в исходном пространстве. Также можно показать, что объекты из того же распределения, из которого сгенерирована обучающая выборка, будут с не очень большой вероятностью попадать в область с отрицательным значением $a(x)$.

0.7 Опишите процедуры обучения и применения для Isolation Forest.

Ранее мы обсуждали, что случайный лес вводит функцию расстояния — чем чаще два объекта попадают в один лист, тем более похожими их можно считать. Похожий подход можно использовать и для обнаружения аномалий. Метод, который мы разберём, называют изоляционным лесом (Isolation forest) **?**.

На этапе обучения будем строить лес, состоящий из N деревьев. Каждое дерево будем строить стандартным жадным алгоритмом, но при этом признак и порог будем выбирать случайно. Строить дерево будем до тех пор, пока в вершине не окажется ровно один объект, либо пока не будет достигнута максимальная высота. Высоту дерева можно ограничить величиной $\log_2 \ell$.

Метод основан на предположении о том, что чем сильнее объект отличается от большинства, тем быстрее он будет отделён от основной выборки с помощью случайных разбиений. Соответственно, выбросами будем считать те объекты, которые оказались на небольшой глубине.

Чтобы вычислить оценку аномальности объекта x, найдём расстояние от соответствующего ему листа до корня в каждом дереве. Если лист, в котором оказался объект, содержит только его, то в качестве оценки $h_n(x)$ от данного n-го дерева будем брать саму глубину k; если же в листе оказалось m объектов, то в качестве оценки возьмём величину $h_n(x) = k + c(m)$. Здесь $c(m)$ — средняя длина пути от корня до листа в бинарном дереве поиска, которая вычисляется по формуле

$$c(m) = 2H(m-1) - 2\frac{m-1}{m},$$

а $H(i) \approx \ln(i) + 0.5772156649$ — i-е гармоническое число. Оценку аномальности вычислим на основе средней глубины, нормированной на среднюю длину пути в дереве, построенном на выборке размера ℓ:

$$a(x) = 2^{-\frac{\frac{1}{N}\sum_{n=1}^{N} h_n(x)}{c(\ell)}}.$$

Для ускорения работы можно строить каждое дерево на подвыборке размера s; в этом случае во всех формулах выше нужно заменить ℓ на s.

0.8 Как устроен наивный байесовский классификатор?

Если мы предполагаем, что вектор параметров θ случайная величина, мы находимся в зоне действия байесовской статистики. В байесовской статистике используются те же самые модели, что и в частотной, но способ оценивания параметров меняется с максимизации правдоподобия на байесовкий вывод.

Так как θ случайная величина, нам надо высказать своё мнение о ней в терминах плотностей распределения. Обычно такое мнение о распределении параметра называют априорным распределением.

Пусть $p(\theta)$ — априорное распределение на векторе параметров θ. В качестве функции правдоподобия для данного вектора возьмем апостериорное распределение на ответах $p(y \mid x, \theta)$. Тогда по формуле Байеса

$$p(\theta \mid y, x) = \frac{p(y \mid x, \theta)p(\theta)}{p(y \mid x)}.$$

После байесовского вывода мы получаем для каждого параметра в качестве оценки целое апостериорное распределение. С помощью него обычно можно найти ответы на все наши вопросы.

Если хочется получить точечную оценку, можно взять моду апостериорного распределения. Такой подход называют *байесом для бедных*. Обычно найти моду апостериорного распределения гораздо проще, чем сделать байесовский вывод.

$$argmax_\theta p(\theta \mid y, x) = argmax_\theta p(y \mid x, \theta)p(\theta) = argmax_\theta [\log p(y \mid x, \theta) + \log p(\theta)]$$

Получается, что при таком подходе мы максимизируем правдоподобие с некоторой добавкой. Эта добавка и представляет из себя регуляризацию.

Метод максимального правдоподобия — частный случай байесовских методов. В нём в качестве априорного распределения на параметры мы берём равномерное, а затем ищем моду апостериорного распределения.

Вернемся к примеру с линейной регрессией. Введем априорное распределение на векторе весов:

$$p(w_j) = \mathcal{N}(0, \alpha^2), \quad j = 1, \ldots, d.$$

Иными словами, мы предполагаем, что веса концентрируются вокруг нуля.

Как было сказано ранее, при применении байесовского классификатора необходимо решить задачу восстановления плотности $p_y(x)$ для каждого класса $y \in \mathbb{Y}$. Данная задача является довольно трудоёмкой и не всегда может быть решена, особенно в случае большого количества признаков, — в частности, если объектами являются тексты, приходится работать с крайне большим числом признаков, и восстановление плотности многомерного распределения не представляется возможным.

Для разрешения этой проблемы сделаем предположение о независимости признаков. В этом случае функция правдоподобия класса y для объекта $x = (x_1, \ldots, x_d)$ может быть представлена в следующем виде:

$$p(x \mid y) = \prod_{j=1}^{d} p(x_j \mid y),$$

где $p(x_j \mid y)$ — одномерная плотность распределения j-ого признака объектов класса $y \in Y$. В этом случае формула байесовского решающего правила примет следующий вид:

$$a(x) = \arg\max_{y \in Y} p(y \mid x) = \arg\max_{y \in \mathbb{Y}} \left(\ln p(y) + \sum_{j=1}^{d} \ln p(x_j \mid y) \right).$$

Предположение о независимости признаков существенно облегчает задачу, поскольку вместо решения задачи восстановления d-мерной плотности необходимо решить d задач восстановления одномерных плотностей. Полученный классификатор называется *наивным байесовским классификатором*.

Плотности отдельных признаков могут быть восстановлены различными способами (параметрическими и непараметрическими). Среди параметрических способов чаще всего используются нормальное распределение (для вещественных признаков), распределение Бернулли и мультиномиальное распределение (для дискретных признаков), благодаря которым получаются различные применяющиеся на практике модели.

0.9 Опишите вероятностную модель, для которой максимизация правдоподобия аналогична обучению линейной регрессии на MSE с L2-регуляризатором.

Покажите, что максимизация апостериорной вероятности $p(w \mid y, x)$ для модели линейной регрессии с нормальным априорным распределением эквивалентна решению задачи гребневой регрессии.

Запишем апостериорную вероятность вектора весов w для выборки $x_1, \ldots, x_\ell$:

$$p(w \mid y, x) = \prod_{i=1}^{\ell} p(y_i \mid x_i, w) p(w) = \\ = \prod_{i=1}^{\ell} \frac{1}{\sqrt{2\pi\sigma^2}} \exp\left(-\frac{(y_i - \langle w, x_i \rangle)^2}{2\sigma^2}\right) \prod_{j=1}^{d} \frac{1}{\sqrt{2\pi\alpha^2}} \exp\left(-\frac{w_j^2}{2\alpha^2}\right).$$

Перейдем к логарифму и избавимся от константных членов:

$$\log p(w \mid y, x) = -\frac{1}{2\sigma^2} \sum_{i=1}^{\ell} (y_i - \langle w, x_i \rangle)^2 - \frac{\ell}{2\alpha^2} \underbrace{\sum_{j=1}^{d} w_j^2}_{=\|w\|^2}.$$

В итоге получаем задачу гребневой регрессии

$$\sum_{i=1}^{\ell} (y_i - \langle w, x_i \rangle)^2 + \lambda \|w\|^2 \to \min_w,$$

где $\lambda = \frac{\ell}{2\alpha^2}$.

После того, как оптимальный вектор весов w_* найден, мы можем найти распределение на ответах для нового объекта x:

$$p(y \mid x, X, w_*) = \mathcal{N}(\langle x, w_* \rangle, \sigma^2).$$

Выше мы выяснили, что оптимальным ответом будет матожидание $(y \mid x) = \int y p(y \mid x, X, w_*) dy$.

С точки зрения байесовского подхода **?** правильнее не искать моду[1] w_* апостериорного распределения на параметрах и брать соответствующую ей модель $p(y \mid x, X, w_*)$, а устроить «взвешенное голосование» всех возможных моделей:

$$p(y \mid x, X) = \int p(y \mid x, w) p(w \mid Y, X) dw,$$

где $X = \{x_1, \ldots, x_\ell\}$, $Y = \{y_1, \ldots, y_\ell\}$.

0.10 Описание алгоритма DBSCAN.

Это алгоритм, основанный на плотности — если дан набор объектов в некотором пространстве, алгоритм группирует вместе объекты, которые расположены близко и помечает как выбросы объекты, которые находятся в областях с малой плотностью (ближайшие соседи которых лежат далеко). Алгоритм имеет два основных гиперпараметра:

- ‘eps‘ радиус рассматриваемой окрестности

- ‘minsamples‘ число соседей в окрестности

Все точки делятся на основные точки, достижимые по плотности точки и выбросы следующим образом:

- Точка p является основной точкой, если по меньшей мере ‘minsamples‘ точек находятся на расстоянии, не превосходящем ‘eps‘ до неё. Говорят, что эти точки достижимы прямо из p.

[1] Мода — точка максимума плотности.

- Точка q прямо достижима из p, если точка q находится на расстоянии, не большем ‘eps‘ , от точки p и p должна быть основной точкой.

- Точка q достижима из p, если имеется путь $p_1, \ldots, p_n$ где $p_1 = p$ и $p_n = q$, а каждая точка p_{i+1} достижима прямо из p_i (все точки на пути должны быть основными, за исключением q) - Все точки, не достижимые из основных точек, считаются выбросами.

- Если p является основной точкой, то она формирует кластер вместе со всеми точками (основными или неосновными), достижимые из этой точки. Каждый кластер содержит по меньшей мере одну основную точку. Неосновные точки могут быть частью кластера, но они формируют его «край», поскольку не могут быть использованы для достижения других точек.

Простыми словами:

1. Находим все основные точки (такие, что в радиусе *eps* от них лежит хотя бы minsamples точек).

2. Находим связные компоненты этих точек на графе соседей и оформляем кластеры.

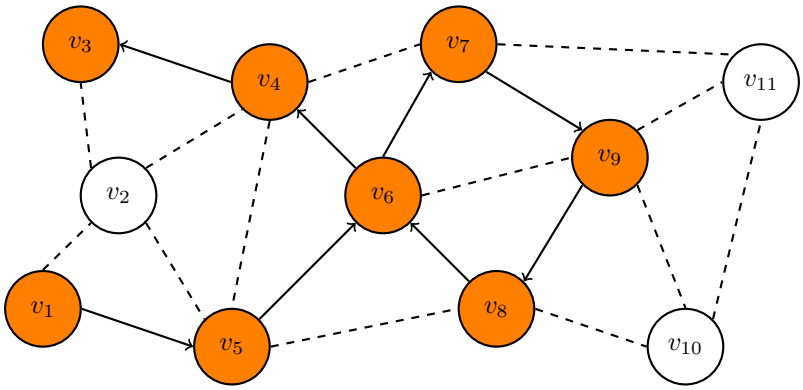
3. Назначаем неосновным точкам ближайший *eps*-соседний кластер, если так можно сделать, в противном случае выкидываем точку в шум.

0.11 Алгоритм node2vec для обучения представлений вершин графа.

Главная особенность метода — использование вероятности случайного блуждания в качестве функции похожести. Пусть вероятность попасть из вершины v_i в вершину v_j при случайном блужданиии равна $p(v_j|v_i)$. Мы хотим обучить такие векторы-эмбеддинги z, что выполняется:

$$p(v_j|v_i) \approx \frac{e^{z_j^T z_i}}{\sum_k e^{z_k^T z_i}} = (z_i, z_j)$$

Иными словами, в качестве декодера мы будем использовать оператор Softmax от скалярных произведений векторных представлений. Здесь возникает следующая проблема: посчитать $p(v_j|v_i)$ аналитически мы можем только для очень простых графов, но для реальных данных это едва ли возможно. Тем не менее, мы можем запустить случайное блуждания по графу и оценить, какие вершины чаще встречаются на одном пути. Фактически, node2vec — это модель *skip-gram word2vec*, в которой словами являются вершины графа, а предложениями — случайные пути в этом графе. Так что если читатель знаком с последней, то следующие абзацы станут для него повторением уже пройденного.



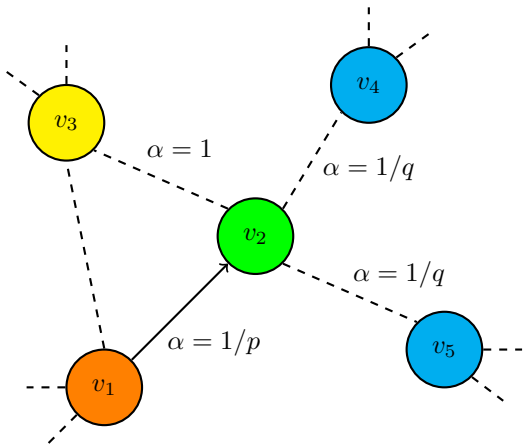
Рассмотрим в качестве примера граф выше и случайный путь W по нему. Пусть $S = [v_1, v_5, v_6, v_7, v_9, v_8, v_6, v_4, v_3]$. Теперь зафиксируем некоторый размер окна w, например, $w = 2$, и для каждой вершины, которая встретилась на пути посмотрим на окно с центром в ней. Для начальной вершины v_1 окно будет выглядеть как $[\underline{v_1}, v_5, v_6]$, для первого вхождения v_6 получим $[v_1, v_5, \underline{v_6}, v_7, v_9]$, а для второго — $[v_9, v_8, \underline{v_6}, v_4, v_3]$. Далее, для каждого окна выпишем пары вида (центр окна, другое упоминание вершины в окне), напимер для окна $[v_1, v_5, \underline{v_6}, v_7, v_9]$ получим пары (v_6, v_1), (v_6, v_5), (v_6, v_7), (v_6, v_9). Назовем такие пары (v_i, v_j) позитивными и постараемся собрать таких как можно

больше, увеличивая длину и число запусков случайного блуждания. Теперь, чтобы обучить векторы-эмбеддинги, мы будем минимизировать следующий функционал:

$$\mathcal{L} = \sum_{(v_i, v_j)} -\log\big((z_i, z_j)\big) = \sum_{(v_i, v_j)} \Big(-z_j^T z_i + \log \sum_k e^{z_k^T z_i} \Big) \to \min_z$$

Единственная проблема, которую осталось решить — подсчет знаменателя в операторе Softmax, который имеет линейную сложность по числу вершин, что будет работать очень медленно для больших графов. Чтобы избежать это недоразумение, пользуются трюком под названием *negative sampling* — вместо подсчета всей суммы экспонент мы можем случайно выбрать несколько негативных пар (то есть вершин, которые не попали в окно с центром в v_i) и с их помощью оценить знаменатель, то есть теперь мы считаем знаменатель за константное время вместо линейного.

Обученные методом node2vec векторные представления будут содержать информацию о близости вершин с точки зрения случайного блуждания. Но случайное блуждание случайному блужданию рознь. Мы можем захотеть, чтобы наши эмбеддинги хорошо описывали локальную окрестность каждой вершины, тогда будет разумно взять случайное блуждание, похожее на обход графа в ширину. Или наоборот, мы можем захотеть пути, которые пробегаются по всему графу, не застревая в одном месте, тогда было бы уместно рассмотреть обход в глубину. Авторы node2vec вводят два гиперпараметра p и q, которые позволяют переключаться между этими двумя режимами. Посмотрим на картинку:



Допустим, на очередном шаге блуждания мы попали из вершины v_1 в вершину v_2. Чтобы определить вероятности перехода в соседние вершины, зададим веса ребрам α:

1. Для ребра, ведущего обратно, положим $\alpha = 1/p$.
2. Для общих соседей предыдущей и текущей вершины положим $\alpha = 1$.
3. Для всех остальных вершин положим $\alpha = 1/q$.

Итоговая вероятность для ребра получаются как его вес, деленный на сумму всех весов ребер, выходящих из v_2. Проанализируем, что у нас получилось. Гиперпараметр p контролирует вероятность возврата в предыдущую вершину. Чем больше p, тем больше мы будем посещать новые вершины вместо возврата обратно. Гиперпараметр q регулирует уклон в сторону обхода в глубину/в ширину. Чем больше q, тем меньше вероятность покинуть соседей v_1, то есть мы больше склонны к обходу в ширину. И наоборот, чем меньше q, тем больше наше блуждание похоже на обход в глубину. Да, такая постановка дает нам два новых гиперпараметра, которые нужно подбирать, но делает модель гораздо более гибкой с точки зрения информации, которая сохраняется в эмбеддингах.

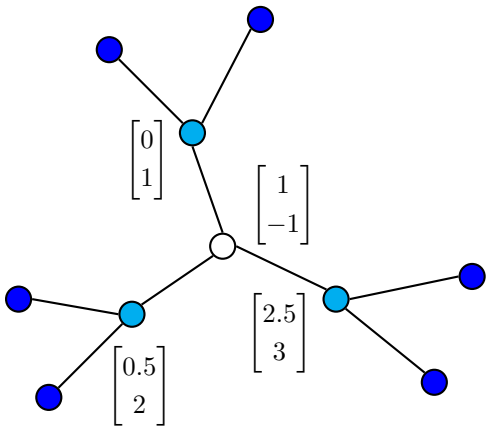
Обсудим теперь, какие плюсы и минусы есть у метода node2vec. Среди достоинств можно выделить гибкость модели: мы можем выбирать, какую именно информацию заложить в эмбеддинги. Кроме того, модель радует

своей простотой: пусть мы и обсудили много подробностей про случайные блуждания на графах, учим-то мы только саму матрицу эмбеддингов. В связи с ней же появляются следующие недостатки: мы не сможем создавать векторные представления для новых вершин, а также у нас нет никаких общих параметров для разных вершин, что может привести к переобучению. Более того, мы не используем метаданные, которые могут содержаться в вершинах. Далее мы поговорим про другие алгоритмы, которые исправляют минусы node2vec.

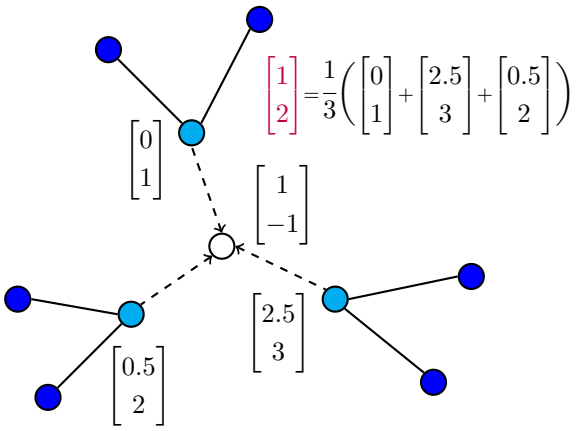
0.12 Алгоритм neighborhood aggregation для обучения представлений вершин графа.

Попробуем придумать такой метод, который будет одновременно использовать структуру графа, учитывать метаданные, а также подойдет для генерации эмбеддингов для новых вершин. Для этого подойдет идея *neighborhood aggregation*: для каждой вершины мы будем агрегировать признаки ее соседей и некоторым образом комбинировать с признаками самой вершины. Более подробно, алгоритм выглядит так:

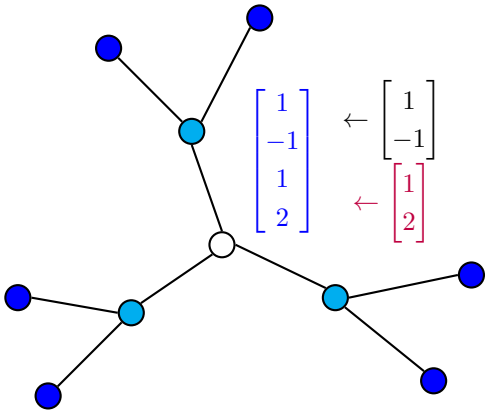
1. Начинаем с того, что у каждой вершины есть некоторый вектор с признаками (исходно можно взять метаданные вершин).
2. Для каждой вершины агрегируем признаки ее соседей (в простейшем случае просто усредняем их).
3. Комбинируем вектор признаков вершины и вектор признаков соседей (в простейшем случае конкатенируем их).
4. К комбинированному вектору применяем линейное преобразование W и нелинейную функцию σ. Получаем новые векторы признаков для каждой вершины.
5. Теперь последовательно повторяем шаги 2–4 K раз (с разными матрицами весов W_k для $k = 1, \dots, K$).



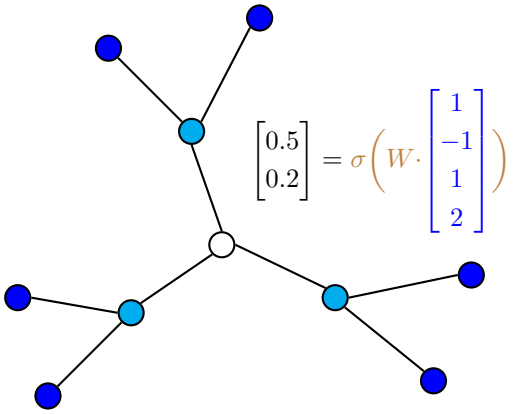
(1) Исходные векторы признаков



(2) Агрегируем признаки соседей



(3) Комбинируем векторы признаков



(4) Вычисляем новые векторы признаков

Фактически, мы получили полносвязную нейронную сеть, в которой связи между нейронами зависят от связей между вершинами в графе. Осталось обсудить, как именно учить такую модель, потому что параметры $\{W_k\}_{k=1}^K$ остаются ненастроенными. Делать это мы будем, разумеется, с помощью градиентного спуска (все использованные нами преобразования были дифференцируемыми), а в качестве функции потерь можно взять любую, из описанных выше, например, приближение некоторой функции похожести вершин или подход со случайными блужданиями, использованный в node2vec.

По сути, neighborhood aggregation — это не что иное, как более продвинутый энкодер для генерации эмбеддингов. Он использует метаданные вершин, а также позволяет обрабатывать новые вершины, которые не встречались в обучающей выборке (мы точно так же можем провернуть шаги алгоритма для новых вершин, просто веса W_k уже будут обученными).

Отдельно стоит упомянуть гиперпараметр глубины K: чем он больше, тем более широкая окрестность вершины влияет на ее векторное представление. За это приходится платить большим временем обработки и возможным переобучением засчет увеличения числа параметров (хотя, гипотетически, никто не запрещает нам брать одинаковые матрицы линейных преобразований на каждом шаге $W_k = W$, тогда у модели будут разделяемые параметры).

0.13 Лапласиан графа, его свойства.

Мы можем представить объекты из выборки в виде вершин некоторого неориентированного графа $G = (\mathcal{V}, \mathcal{E})$, $\mathcal{V} = X = \{x_1, \ldots, x_l\}$. Рассмотрим несколько вариантов, как в таком представлении можно задать ребра $\mathcal{E}$:

1. Граф G может быть полным с ребрами, вес которых определяется по некоторой формуле, например:

$$w_{ij} = \exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma^2}\right)$$

   Гиперпараметр σ определяет, насколько нам важны далекие объекты.

2. G можно задать как kNN-граф, то есть объект x_i будет связан с k его ближайшими соседями.

3. Вершина x_i может быть связана с теми вершинами, расстояние до которых меньше выбранного ϵ, то есть $\rho(x_i, x_j) < \epsilon$. Такой граф будет называться ϵ-графом.

Лапласиан графа

Обозначим через W матрицу смежности графа G. Степени вершин будем считать как $d_i = \sum_{j=1}^l w_{ij}$. Пусть $D = (d_1, \ldots, d_l)$. Тогда матрица $L = D - W$ будет называться *лапласианом* графа G. Рассмотрим несколько свойств лапласиана L:

1. Пусть $f \in \mathbb{R}^n$. Тогда имеет место следующая формула:

$$f^\intercal L f = \frac{1}{2} \sum_{i,j=1}^{l} w_{ij}(f_i - f_j)^2$$

Проверка формулы:

$$\begin{aligned}
f^\intercal L f = f^\intercal D f - f^\intercal W f = \sum_{i=1}^{l} d_i f_i^2 - \sum_{i,j=1}^{l} w_{ij} f_i f_j = \\
= \sum_{i=1}^{l} \left(\sum_{j=1}^{l} w_{ij} \right) f_i^2 - \sum_{i,j=1}^{l} w_{ij} f_i f_j = \sum_{i,j=1}^{l} w_{ij} f_i^2 - \sum_{i,j=1}^{l} w_{ij} f_i f_j = \\
= \frac{1}{2} \sum_{i,j=1}^{l} w_{ij} f_i^2 + \frac{1}{2} \sum_{i,j=1}^{l} w_{ij} f_i^2 - \sum_{i,j=1}^{l} w_{ij} f_i f_j = \\
= \frac{1}{2} \sum_{i,j=1}^{l} w_{ij} f_i^2 + \frac{1}{2} \sum_{i,j=1}^{l} w_{ij} f_j^2 - \sum_{i,j=1}^{l} w_{ij} f_i f_j = \\
= \frac{1}{2} \sum_{i,j=1}^{l} w_{ij} \left(f_i^2 - 2 f_i f_j + f_j^2 \right) = \frac{1}{2} \sum_{i,j=1}^{l} w_{ij} (f_i - f_j)^2
\end{aligned}$$

2. L — симметричная неотрицательно определенная матрица. Симметричность вытекает из неориентированности графа. Свойство неотрицательной определенности легко следует из первого пункта. Действительно, в обсужденных методах постоения графа $w_{ij} \geq 0$, притом $(f_i - f_j)^2 \geq 0$. Следовательно, $f^\intercal L f \geq 0$, что и означает неотрицательную определенность.

Однако у лапласиана также есть свойство, которое поможет нам в задаче кластеризации. Сформулируем его в виде теоремы.

Пусть L — лапласиан графа G. Тогда выполнены следующие два пункта:

1. Собственное значение $\lambda = 0$ матрицы L имеет кратность, равную числу компонент связности k.

2. Пусть $A_1, \ldots, A_k$ — компоненты связности графа G. Тогда векторы $f_1, \ldots, f_k$, определяемые по формуле $f_i = ([x_j \in A_i])_{j=1}^{l}$, будут являться собственными векторами для $\lambda = 0$.

Сперва рассмотрим случай $k = 1$. Поймем, почему $\lambda = 0$ вообще является собственным значением матрицы L. Для этого рассмотрим вектор $f = (1, \ldots, 1)$:

$$\begin{aligned}
Lf = Df - Wf = \begin{pmatrix} d_1 & 0 & \cdots & 0 \\ 0 & d_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & d_l \end{pmatrix} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} - \begin{pmatrix} w_{11} & w_{12} & \cdots & w_{1l} \\ w_{21} & w_{22} & \cdots & w_{2l} \\ \vdots & \vdots & \ddots & \vdots \\ w_{l1} & w_{l2} & \cdots & w_{ll} \end{pmatrix} \begin{pmatrix} 1 \\ \vdots \\ 1 \end{pmatrix} = \\
= \begin{pmatrix} d_1 \\ \vdots \\ d_l \end{pmatrix} - \begin{pmatrix} w_{11} + \cdots + w_{1l} \\ \vdots \\ w_{l1} + \cdots + w_{ll} \end{pmatrix} = 0
\end{aligned}$$

Теперь предположим, что существует собственный вектор $f' \in \mathbb{R}^n : \exists p \neq q \to f'_p \neq f'_q$, то есть неконстантный вектор, соответствующий $\lambda = 0$. Тогда $Lf' = 0 \Rightarrow f'^\intercal L f' = 0$. Но рассматриваемый граф является связным. Значит существует путь $p = i_0 \to i_1 \to \cdots \to i_{n-1} \to i_n = q$. Поскольку вершины i_r и i_{r+1} соединены ребром, то $w_{i_r i_{r+1}} > 0$, а значит, $f'_{i_r} = f'_{i_{r+1}}$ (иначе получим $f'^\intercal L f' > 0$). Отсюда $f'_p = f'_{i_1} = \cdots = f'_{i_{n-1}} = f'_q$ - константный

вектор. Получили противоречие $\Rightarrow$ f' не является собственным вектором для $\lambda = 0$. Значит, мы доказали оба пункта для $k = 1$.

Теперь пусть $k > 1$. Можно упорядочить вершины так, чтобы лапласиан L стал блочно-диагональной матрицей:

$$L = \begin{pmatrix} L_1 & 0 & \cdots & 0 \\ 0 & L_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & L_k \end{pmatrix}$$

Блоки $L_1, \ldots, L_k$ будут являться лапласианами компонент связности графа G $\Rightarrow$ $\lambda = 0$ имеет кратность k, а собственные векторы $f_1, \ldots, f_k$ задаются по формуле

$$f_i = ([x_j \in A_i])_{j=1}^{l}$$

Пусть есть похожие объекты x_j и x_k, то есть расстояние между ними невелико. Тогда у собственных векторов f_i, соответствующих маленьким собственным значениям, выполнено $f_{ij} \approx f_{ik}$.

Эта гипотеза имеет лишь эмпирическое доказательство. Однако алгоритм, построенный на этой гипотезе, позволяет достигать успеха в задаче кластеризации. Данный алгоритм называется *спектральной кластеризацией*.

0.14 Алгоритм спектральной кластеризации.

Алгоритм спектральной кластеризации: *Сложность шага*

1. Строим по объектам граф G и лапласиан $L = D - W$. $\mathcal{O}(l^2)$
2. Находим нормированные собственные векторы $u_1, \ldots, u_m$ матрицы L, соответствующие m наименьшим собственным значениям. $\mathcal{O}(l^3)$
3. Составляем матрицу $U \in \mathbb{R}^{l \times m}$: $U = (u_1 | \ldots | u_m)$. $\mathcal{O}(lm)$
4. Обучаем на этой матрице алгоритм K-Means с k кластерами. $\mathcal{O}(l^{mk+1})$

Если мы верим в нашу гипотезу, то для похожих объектов соответствующие координаты векторов $u_1, \ldots, u_m$ будут близки. Матрица U имеет размерность $l \times m$, поэтому можно посмотреть на нее, как на матрицу объекты-признаки. При этом новыми признаками будут как раз собственные вектора $u_1, \ldots, u_m$. В описанном выше алгоритме предлагается обучать на них K-Means, но никто не запрещает использовать эти признаки для других задач.

0.15 Как считается метрика BCubed? Чем она хороша?

Введем несколько требований к внешней метрике качества Q:

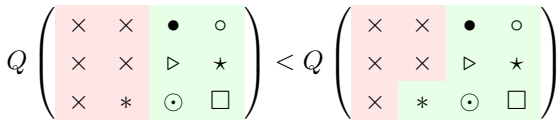
1. *Гомогенность*. Базовое свойство разделения разных объектов в разные кластеры:

$$Q\left(\begin{array}{ccc} & \diamond & \diamond \\ \times & & \diamond \\ \times & \times & \end{array}\right) < Q\left(\begin{array}{ccc} & \diamond & \diamond \\ \times & & \diamond \\ \times & \times & \end{array}\right)$$

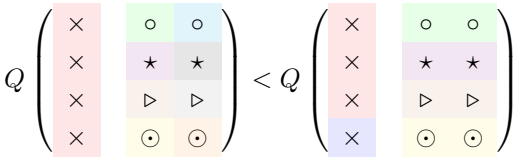
2. *Полнота*. Один кластер не должен дробиться на несколько маленьких:

$$Q\left(\begin{array}{ccc} & \times & \times \\ \times & & \times \\ \times & \times & \end{array}\right) < Q\left(\begin{array}{ccc} & \times & \times \\ \times & & \times \\ \times & \times & \end{array}\right)$$

3. *Rag-bag.* Весь мусор должен быть в одном "мусорном"кластере, чтобы остальные кластеры были "чистыми":



4. *Cluster size vs. quantity.* Лучше испортить один кластер с целью улучшить качество множества других:



BCubed

Единственной известной на данный момент метрикой, обладающей всеми четыремя названными свойствами является BCubed. Она считается следующим способом. Пусть $L(x)$ — gold standard, $C(x)$ — номер кластера, выдаваемый рассматриваемым алгоритмом. Тогда рассмотрим несколько величин:

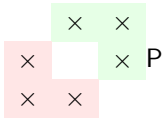
$$\mathsf{Correctness}(x, x') = \begin{cases} 1 & , \ C(x) = C(x') \wedge L(x) = L(x') \\ 0 & , \ otherwise \end{cases}$$

$$\mathsf{Precision\text{-}BCubed} = \mathsf{Avg}_x \left[\mathsf{Avg}_{x': C(x)=C(x')} \mathsf{Correctness}(x, x') \right]$$

$$\mathsf{Recall\text{-}BCubed} = \mathsf{Avg}_x \left[\mathsf{Avg}_{x': L(x)=L(x')} \mathsf{Correctness}(x, x') \right]$$

Тогда F-мера от определенных точности и полноты будет удволетворять всем нужным нам требованиям.

$$Fmeasure_\beta = \frac{(1+\beta^2) Precision \cdot Recall}{\beta^2 Precision + Recall}$$

 $\mathsf{Precision\text{-}BCubed} = 1, \mathsf{Recall\text{-}BCubed} = \frac{1}{2}, \mathsf{BCubed} = \frac{1 \cdot \frac{1}{2}}{1 + \frac{1}{2}} = \frac{1}{3}$

0.16 Опишите метод PLSA для тематического моделирования.

Будем считать, что каждый документ x_d описывается распределением $p(t \mid d) = \theta_{td}$, а каждая тема — распределением $p(w \mid t) = \phi_{wt}$. Тогда совместное распределение на словах и документах можно записать как

$$p(w, d) = p(d)p(w \mid d) = p(d) \sum_{t=1}^{T} p(w \mid t)p(t \mid d).$$

Здесь мы, по сути, ввели скрытую переменную t, которая показывает, из какой темы было сгенерировано слово w документа x_d. Согласно данной модели, документ x_d генерируется по следующей схеме:

1. Выбираем тему $t \sim p(t \mid d)$;
2. Выбираем слово из данной темы $w \sim p(w \mid t)$;
3. Повторяем шаги 1 и 2, если текст не достиг требуемой длины.

Чтобы записать правдоподобие, следует смотреть на набор документов как на выборку пар «документ-слово». Неполное правдоподобие данной модели имеет вид

$$\sum_{d=1}^{D}\sum_{j=1}^{|x_d|}\log\sum_{t=1}^{T}\phi_{w_{dj}t}\theta_{td},$$

где w_{dj} — j-е по порядку слово из документа x_d. Если для каждой пары «документ-слово» (d, w_{dj}) известно, из какой темы t_{dj} оно сгенерировано, то можно записать полное правдоподобие:

$$\sum_{d=1}^{D}\sum_{j=1}^{|x_d|}\sum_{t=1}^{T}[t_{dj}=t]\log\phi_{w_{dj}t}\theta_{td}.$$

Мы уже знаем, что для обучения таких моделей можно воспользоваться EM-алгоритмом. На E-шаге оценим апостериорные распределения на скрытых переменных по формуле Байеса:

$$p(t_{dj}\mid d, w_{dj}) = \frac{p(w_{dj}\mid t_{dj})p(t_{dj}\mid d)}{p(w_{dj}\mid d)} = \frac{\phi_{w_{dj}t_{dj}}\theta_{t_{dj}d}}{p(w_{dj}\mid d)}$$

На M-шаге найдём максимум матожидания полного правдоподобия по скрытым переменным:

$$\phi_{wt} = \frac{\sum_{d=1}^{D} n_{dw}p(t\mid d,w)}{\sum_{w=1}^{W}\sum_{d=1}^{D} n_{dw}p(t\mid d,w)};$$
$$\theta_{td} = \frac{\sum_{w=1}^{W} n_{dw}p(t\mid d,w)}{\sum_{t=1}^{T}\sum_{w=1}^{W} n_{dw}p(t\mid d,w)}.$$

Здесь n_{dw} — число вхождений слова w в документ x_d.

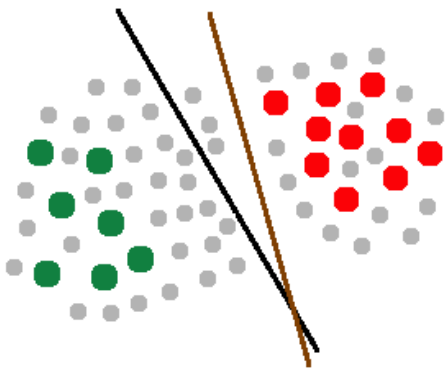
Полученная в итоге работы EM-алгоритма модель будет интепретируемой — можно изучать, насколько сильно та или иная тема представлена в документе, или насколько то или иное слово характерно для темы.

Модель PLSA не является полной — распределения ϕ_t и θ_d считаются параметрами, не предполагается, что они тоже генерируются из некоторых распределений. Значит, с помощью данной модели не получится описать процесс порождения набора документов «с нуля». Более того, в PLSA отсутствует регуляризация, из-за чего модель может слишком сильно подогнаться под данные на небольших выборках.

0.17 Как устроен метод self-training?

Background

Есть размеченная обучающая выборка $X^\ell = \{(x_i, y_i)\}_{i=1}^{\ell}$ и неразмеченная обучающая выборка $X^u = \{x_i\}_{i=\ell+1}^{n}$.



Метод Self-training

Рассмотрим следующий алгоритм:

1. Обучить модель $a(x)$ на X^ℓ
2. Применить $a(x)$ на X^u
3. Добавить $(x_i, a(x_i))$ к X^ℓ для некоторых $x_i \in X^u$
4. Вернуться к шагу 1.

В пункте 3, в случае классификации, можно например добавлять объекты, на которых модель имеет наибольшую уверенность. Также можно добавлять объекты, близкие к X^ℓ. При этом просто добавить все объекты из X^u может быть плохой идеей, так как если в неразмеченной выборке есть выбросы, мы только подпортим себе жизнь. Помочь бороться с этой проблемой может добавление в целевой функционал весов, равных уверенностям модели в предсказаниях на X^u.

0.18 Как устроен метод semi-supervised SVM?

Вариант безусловной задачи оптимизации обычного SVM с hinge loss

$$\|w\|^2 + C\sum_{i=1}^{\ell} \max(0, 1 - y_i\langle w, x_i\rangle) \to \min_w$$

Неразмеченные объекты не должны находиться слишком близко к разделяющей гиперплоскости.

$$\|w\|^2 + C_1\sum_{i=1}^{\ell} \max(0, 1 - y_i\langle w, x_i\rangle) + C_2\sum_{i=\ell+1}^{n} \max(0, 1 - |\langle w, x_i\rangle|) \to \min_w.$$

То есть штрафовать мы будем за неразмеченные объекты с модулем отступа меньше 1.

Но может оказаться, что из-за второго слагаемого оптимальной с точки зрения нашего функционала гиперплоскостью будет такая, что вообще все объекты лежат только по одну сторону от нее. Такая ситуация нас по понятным причинам не устроит.

Баланс предсказанных классов на неразмеченной выборке должен быть таким же, как и баланс классов на размеченной выборке по исходным меткам.

$$\frac{1}{n-\ell}\sum_{i=\ell+1}^{n} \langle w, x_i\rangle = \frac{1}{\ell}\sum_{i=1}^{\ell} y_i.$$

В левой части мы по хорошему должны были брать $sign(\langle w, x_i\rangle)$, но тогда бы вышла слишком сложная оптимизационная задача, поэтому мы разрешаем себе такое послабление.

Итоговую задачу можно решать методом оптимизации под названием CCCP (ConCave-Convex Procedure).

0.19 В чём заключается регуляризация на основе лапласиана графа в частичном обучении?

Давайте скажем, что на размеченных объектах мы хотим получать просто близкие к верным предсказания. Чтобы учитывать и неразмеченные объекты, давайте дополнительно потребуем, что если два объекта выборки близки, мы должны также выдавать на них похожие предсказания. Тогда может получиться что-то такое:

$$\sum_{i=1}^{\ell} L(y_i, a(x_i)) + \lambda_1 R(a) + \lambda_2\sum_{i,j=1}^{n} w_{i,j}(a(x_i) - a(x_j))^2 \to \min_a.$$

Здесь L — функция потерь, R — регуляризатор, а $w_{i,j}$ например $\exp(-\frac{\|x_i - x_j\|^2}{2\sigma^2})$. Можно вспомнить, что последнее слагаемое тогда является одним из вариантов записи Лапласиана $a^T La$, где $a = (a(x_1), ..., a(x_n))^T$. Данный подход носит название manifold regularization.

0.20 Как работает метод k ближайших соседей для классификации и для регрессии?

Классификация

Задача классификации: $\mathbb{Y} = \{1, \ldots, K\}$. Дана обучающая выборка $X = (x_i, y_i)_{i=1}^{\ell}$ и функция расстояния $\rho : \mathbb{X}\times \to [0, \infty)$.
Функция расстояния не обязательно метрика — достаточно, чтобы она была симметричной и неотрицательной. Не будем строить модель на этапе обучения, а вместо этого просто запомним обучающую выборку.
Пусть теперь требуется классифицировать новый объект u. Расположим объекты обучающей выборки X в порядке неубывания расстояний до u:

$$\rho(u, x_u^{(1)}) \leq \rho(u, x_u^{(2)}) \leq \cdots \leq \rho(u, x_u^{(\ell)}),$$

где через $x_u^{(i)}$ обозначается i-й сосед объекта u. Алгоритм k *ближайших соседей* (k nearest neighbours, kNN) относит объект u к тому классу, представителей которого окажется больше всего среди k его ближайших соседей:

$$a(u) = \underset{y\in\mathbb{Y}}{\operatorname{argmax}} \sum_{i=1}^{k} [y_u^{(i)} = y]. \tag{7}$$

Проблема формулы (7) состоит в том, что она никак не учитывает расстояния до соседей. Поэтому итоговая формула исходит из метода парзеновского окна:

$$a(u) = \underset{y\in\mathbb{Y}}{\operatorname{argmax}} \sum_{i=1}^{k} K\left(\frac{\rho(u, x_u^{(i)})}{h}\right) [y_u^{(i)} = y], \tag{8}$$

где K — это ядро, h — ширина окна.

Регрессия

В методе k ближайших соседей (8) мы, по сути, максимизировали взвешенную долю правильных ответов среди соседей при предсказании константой y. Иными словами, мы старались в каждой точке пространства выбрать такое значение модели, которое было бы оптимально для некоторой окрестности этой точки.

Попробуем воспользоваться этим соображением, чтобы обобщить подход на задачу регрессии. Будем выбирать в каждой точке такой ответ, который лучшим образом приближает целевую переменную для k ближайших соседей:

$$a(u) = \underset{c\in\mathbb{R}}{\operatorname{argmin}} \sum_{i=1}^{k} K\left(\frac{\rho(u, x_u^{(i)})}{h}\right) (c - y_u^{(i)})^2.$$

Можно в явном виде выписать решение оптимизационной задачи:

$$a(u) = \frac{\sum_{i=1}^{k} K\left(\frac{\rho(u,x_u^{(i)})}{h}\right) y_u^{(i)}}{\sum_{i=1}^{k} K\left(\frac{\rho(u,x_u^{(i)})}{h}\right)} \tag{9}$$

Данную модель иногда называют формулой Надарая-Ватсона.

0.21 Что такое расстояние Махалонобиса? Как матрица в нём влияет на линии уровня?

Расстояние Махалонобиса определяется следующим образом:

$$\rho(x, z) = \sqrt{(x - z)^T S^{-1} (x - z)},$$

где S — симметричная положительно определенная матрица.

Матрица и линии уровня Что представляют из себя линии уровня функции $f(x) = \rho(x, 0)$?

Поскольку матрица S — симметричная, то из её собственных векторов можно составить ортонормированный базис. Рассмотрим матрицу Q, столбцами которой являются элементы данного базиса. Заметим, что она является ортогональной (в силу ортонормированности базиса), т.е. $Q^T Q = I$, $Q^{-1} = Q^T$, а потому выполняются следующие соотношения:

$$SQ = Q\Lambda \Rightarrow \Lambda = Q^{-1}SQ,$$

где Λ — диагональная матрица, в которой записаны соответствующие собственные значения матрицы S.

Распишем квадрат функции $f(x)$, сделав замену $x' = Q^T x$:

$$f^2(x) = \rho^2(x, 0) = x^T S^{-1} x = x'^T Q^T S^{-1} Q x' = x'^T \left(Q^{-1} S Q\right)^{-1} x' =$$
$$= x'^T \Lambda^{-1} x' = \sum_{j=1}^{d} \frac{x'^2}{\lambda_j}.$$

Получаем, что линии уровня в новых координатах представляют собой эллипсы с осями, параллельными осям координат, причем длины полуосей равны корням из собственных значений матрицы S. При этом замена $x' = Q^T x$ соответствует такому повороту осей координат, что координатные оси совпадают со столбцами матрицы Q. Таким образом, расстояние Махалонобиса позволяет получить линии уровня в виде произвольно ориентированных эллипсов.

Матрицу S можно настраивать либо по кросс-валидации, либо брать равной выборочной ковариационной матрице: $\hat{S} = \frac{1}{n-1} X^T X$.

0.22 Опишите алгоритм LSH. Как устроены хэши для евклидовой метрики и для косинусного расстояния? Какие параметры есть в композициях хэш-функций и на что они влияют?

Background Есть два способа борьбы с высокой сложностью поиска ближайших соседей при большом числе признаков:

1. Запоминать не всю обучающую выборку, а лишь ее представительное подмножество.
2. Искать k ближайших соседей приближенно, то есть разрешать результату поиска быть чуть дальше от нового объекта, чем k его истинных соседей. Ниже мы подробно разберем этот подход.

Алгоритм LSH Идея заключается в построении такой хэш-функции для объектов выборки, которая с большой вероятностью присваивает одинаковые значения близким объектам и разные значения отдаленным объектам.

Формальное определение. Семейство функций $\mathcal{F}$ называется (d_1, d_2, p_1, p_2)-чувствительным, если для всех $x, y \in \mathbb{X}$ выполнено:

- Если $\rho(x, y) \leq d_1$, то $\mathbb{P}_{f \in \mathcal{F}} [f(x) = f(y)] \geq p_1$.
- Если $\rho(x, y) \geq d_2$, то $\mathbb{P}_{f \in \mathcal{F}} [f(x) = f(y)] \leq p_2$.

Здесь под вероятностью $\mathbb{P}_{f \in \mathcal{F}}$ понимается равномерное распределение на всех функциях семейства $\mathcal{F}$.

Хэш-функции для косинусного расстояния. Для косинусного расстояния используют следующее семейство функций:

$$\mathcal{F} = \{f_w(x) = sign\langle w, x\rangle \mid w \in \mathbb{R}^d\}.$$

Каждая хэш-функция соответствует некоторой гиперплоскости, проходящей через начало координат, и возвращает для каждого вектора либо $+1$, либо -1 в зависимости от того, по какую сторону от этой гиперплоскости он находится.

Хэш-функции для евклидовой метрики. В данном случае хэш-функция соответствует некоторой прямой в d-мерном пространстве, разбитой на отрезки длины r. Функция проецирует объект x на эту прямую и возвращает номер отрезка, в который попала проекция. Формально, семейство хэш-функций имеет вид

$$= \left\{ f_{w,b}(x) = \left\lfloor \frac{\langle w, x \rangle + b}{r} \right\rfloor \mid w \in^d, b \in [0, r) \right\}.$$

При этом, в отличие от описанных выше семейств, функции выбираются не равномерно: каждая компонента проекционного вектора w выбирается из стандартного нормального распределения $\mathcal{N}(0, 1)$.

Данное семейство может быть обобщено на расстояния Минковского с $p \in (0, 2]$. В этом случае компоненты вектора w должны выбираться из p-устойчивого распределения **?**. Например, для $p = 1$ таковым является распределение Коши.

Композиция хэш-функций. Концепция Семейство хэш-функций уже можно использовать для поиска ближайших соседей. Выберем случайную хэш-функцию f, создадим таблицу T, и разместим каждый объект обучающей выборки x в ячейке $f(x)$ этой хэш-таблицы [2]. Пусть теперь требуется найти k ближайших соседей для объекта u. Вычислим для него хэш $f(u)$, возьмем все объекты из соответствующей ячейки хэш-таблицы, и вернем из них k ближайших к u. Однако, как правило, разница между вероятностями p_1 и p_2 оказывается не очень большой, поэтому либо истинные k ближайших соседей не окажутся в ячейке $f(u)$, и результат будет далек от оптимального, либо в эту ячейку попадет слишком много лишних объектов, и тогда поиск окажется слишком трудозатратным.

Чтобы увеличить разницу между вероятностями p_1 и p_2, можно объединять несколько простых хэш-функций из семейства в одну сложную. Выберем для этого m функций $f_1, \ldots, f_m$ из и построим новую функцию $g_1(x) = (f_1(x), \ldots, f_m(x))$. Повторим процедуру L раз и получим L таких функций $g_1(x), \ldots, g_L(x)$. Для каждой функции $g_i(x)$ создадим свою хэш-таблицу T_i, и поместим каждый объект обучающей выборки x в ячейку $g_i(x)$ этой таблицы. Чтобы найти k ближайших соседей для нового объекта u, выберем объекты из ячеек $g_1(x), \ldots, g_L(x)$ таблиц $T_1, \ldots, T_L$ соответственно, и вернем k наиболее близких из них.

Композиция хэш-функций. Параметры Данный алгоритм имеет два параметра: число базовых функций в одной композиции m, и число таких композиций L. Увеличение параметра m приводит к уменьшению вероятности того, что два непохожих объекта будут признаны схожими. Действительно, для того, чтобы значения композиции совпали на двух объектах, необходимо, чтобы совпали значения m базовых хэш-функций. Если расстояние между этими объектами велико, т.е. $\rho(x, y) \geq d_2$, то вероятность совпадения значений m базовых функций не будет превышать p_2^m. В то же время чрезмерное увеличение параметра m может привести к тому, что практически все объекты попадут в разные ячейки хэш-таблицы, и для новых объектов не будет находится ни одного соседа.

Увеличение же параметра L приводит к увеличению вероятности того, что два схожих объекта будут действительно признаны схожими. Действительно, объект x будет рассмотрен нашим алгоритмом как кандидат в k ближайших соседей для u, если хотя бы один из хэшей $g_1(x), \ldots, g_L(x)$ совпадет с хэшем $g_1(u), \ldots, g_L(u)$ соответственно. Если объекты x и u действительно схожи, то есть $\rho(x, u) \leq d_1$, то вероятность того, что они будут признаны схожими, больше или равна $1 - (1 - p_1)^L$ (в случае $m = 1$). В то же время чрезмерное увеличение параметра L приведет к тому, что для нового объекта будет рассматриваться слишком много кандидатов в k ближайших соседей, что приведет к снижению эффективности алгоритма.

0.23 Опишите метод поиска ближайших соседей Product Quantization.

Допустим у нас есть выборка $X = (x_i)_{i=1}^l$, $x_i \in \mathbb{R}^d$. Координаты каждого вектора разбиваем на m блоков (всего $\frac{d}{m}$). Таким образом, каждый вектор разбивается на подвекторы $x_i = (x_i^1, x_i^2, ..., x_i^m), x_i^j \in \mathbb{R}^{\frac{d}{m}}$. Для каждого из m

[2] Поскольку множество значений хэш-функции может быть большим, обычно таблица T сама является хэш-таблицей.

блоков делаем следующее: собираем все вектора, которые встречались в этом блоке по всей обучающей выборке и запускаем кластеризацию (обычно это K-Means, а количество центров 256). Затем для всех векторов, встречающихся в этом блоке, находим ближайший центр кластеризации. Итого мы следующим образом кодируем векторы из обучающей выборки: $x_i = (x_i^1, x_i^2, ..., x_i^m) = (r^1(x_i^1), r^2(x_i^2), ..., r^m(x_i^m))$, где $r^i(x_j^i)$ сопоставляет подвектору i-го блока вектора x_j ближайший центр в кластеризации этого блока. То есть мы кодируем векторы через ближайшие центры кластеров. Тогда для нового вектора q можно будет быстро посчитать расстояния до всех остальных векторов из обучающей выборки:

$$x_i \in X, ||q - x_i||_2^2 \approx ||q - (r^1(x_i^1), r^2(x_i^2), ..., r^m(x_i^m))||_2^2 = \sum_{j=1}^{m} ||q^j - r^j(x_i^j)||_2^2$$

Так как $r^j(x_i^j)$ - центров различных кластеров ограничено, тогда можно будет предпосчитывать все такие расстояния в блоках, следовательно, эффективно считать расстояния до векторов и находить соседей.

0.24 Опишите алгоритмы поиска ближайших соседей NSW и HNSW.

0.24.1 NSW (Navigable small world)

Идея метода: построим граф на $x \in X$, удовлетворяющий условиям

1. Степени вершин графа небольшие;
2. Расстояние между любыми двумя вершинами в среднем небольшое (свойство малого мира в графе);

Теперь, если такой граф построен, то для нового объекта мы ищем ближайшего соседа по следующей схеме:

1. Выбираем случайную вершину в графе;
2. Проверяем, есть ли среди её соседей вершина, которая ближе к новому объекту, чем текущая;
3. Если такой вершины нет, то мы нашли ближайшего соседа, если такая вершина есть — идём в неё и повторяем;

Обозначим за $N(x)$ множество всех соседей вершины x. Описание процедуры GreedSearch поиска ближайшего соседа к новому объекту u, начиная с некоторой вершины x_0:

1. Считаем метрику между текущей вершиной x_0 и u: $\rho_{best} = \rho(x_0, u)$;
2. Находим соседа x_0 с минимальным расстоянием до u: $\rho_{new} = \min\limits_{x \in N(x)} \rho(x, u)$;
3. Если $\rho_{new} < \rho_{best}$, тогда обновляем значение x_0: $x_0 = \arg\min\limits_{x \in N(x)} \rho(x, u)$;
4. Повторяем шаги 2-3, пока объекты x_0 не повторяются;

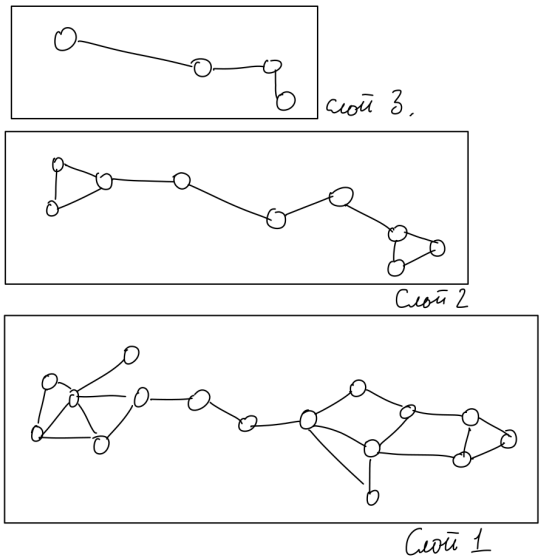
Мы описали процедуру поиска ближайшего соседа в случае, когда имеется некоторый граф. Теперь опишем, как этот граф получить: пусть изначально $G(V, E) = G = \varnothing$. Тогда для каждого объекта $x_i \in X$ выполняем процедуру AddNode(x_i, m, k):

1. Положим $C = \varnothing$, $V = V \cup \{x_i\}$;
2. m раз повторяем следующее: выберем x_0 как случайный элемент выборки $X \setminus V$, и положим в C ближайшего соседа к x_i: $C = C \cup \{GreedSearch(x_i, x_0)\}$;
3. Пополним множество рёбер графа, , добавив в E рёбра между x_i и его k ближайшими соседями из C;

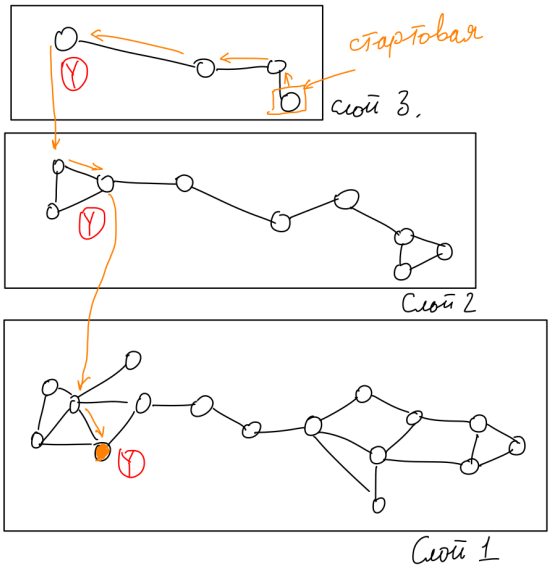
Теперь на этапе применения модели для поиска ближайшего соседа к новому объекту мы можем несколько раз вызвать GreedSearch и вернуть ближайший найденный объект.

0.24.2 HNSW

Пусть имеется граф на X, построенный с помощью NSW. Построим многослойную модель по следующему принципу: пусть на нижнем слое находится исходный граф. На следующем слое мы построим граф с помощью NSW на некотором подмножестве всех вершин, которое мы семплируем равномерно. На следующем слое снова построим граф из некоторого подмножества вершин графа на предыдущем слое, и так далее. В итоге получится примерно такая картинка:



Теперь если нам нужно найти ближайший объект к некоторому новому объекту Y, то мы действуем так: сначала запускаем GreedSearch для графа на самом верхнем слое. Как только найден ближайший сосед, опускаемся на уровень ниже, и ищем ближайшего соседа в графе на следующем слое. И делаем так до тех пор, пока не найдём ближайшего соседа для графа на самом нижнем слое (т.е. ближайшего соседа среди всех вершин). Графически это выглядит так:



0.25 Опишите методы обучения метрик NCA и LMNN.

0.25.1 Neighbourhood Components Analysis

Метод NCA **?** выбирает метрику так, чтобы для каждого объекта ближайшими оказывались объекты его же класса. Рассмотрим объект x_i и рассмотрим следующий эксперимент: мы выбираем из оставшейся выборки случайный объект x_j и относим x_i к классу y_j. Зададим вероятности через расстояния между объектами:

$$p_{ij} = \begin{cases} \frac{\exp(-\|Ax_i - Ax_j\|^2)}{\sum_{k \neq i} \exp(-\|Ax_i - Ax_k\|^2)}, & i \neq j \\ 0, & i = j \end{cases}$$

Можно вычислить вероятность того, что объект x_i будет отнесён к правильному классу. Если обозначить через $C_i = \{j \mid y_i = y_j\}$ множество индексов объектов того же класса, то данная вероятность равна

$$p_i = \sum_{j \in C_i} p_{ij}.$$

Будем максимизировать матожидание количества верно классифицированных объектов:

$$Q(A) = \sum_{i=1}^{\ell} p_i \to \max_A$$

Этот функционал можно продифференцировать по A:

$$\frac{\partial Q}{\partial A} = 2A \sum_i \left(p_i \sum_k p_{ik}(x_i - x_k)(x_i - x_k)^T - \sum_{j \in C_i} p_{ij}(x_i - x_j)(x_i - x_j)^T \right).$$

Далее матрицу A можно обучать любым градиентным методом.

Отметим, что метод NCA можно использовать и для ускорения поиска ближайших соседей. Если взять матрицу $A \in^{n \times d}$ с небольшой первой размерностью n, то она будет переводить объекты в компактные представления, евклидова метрика на которых позволяет хорошо отделять классы друг от друга.

0.25.2 Large margin nearest neighbor

Метод LMNN **?** пытается обучить метрику так, чтобы k ближайших соседей каждого объекта относились к нужному классу, а объекты из других классов отделялись с большим отступом. Попытаемся ввести соответствующий функционал.

Определим для каждого объекта x_i набор из k целевых соседей — объектов, расстояние до которых должно оказаться минимальным. В простейшем варианте это могут быть ближайшие k объектов из этого же класса, но можно выбирать их и иначе. Введём индикатор $\eta_{ij} \in \{0, 1\}$, который равен единице, если объект x_j является целевым соседом для x_i.

Выше мы поставили перед собой две цели: минимизировать расстояние до целевых соседей и максимизировать расстояние до объектов других классов. Суммарное расстояние до целевых соседей можно вычислить как

$$\sum_{i \neq j} \eta_{ij} \|Ax_i - Ax_j\|^2.$$

Для объектов других классов будем требовать, чтобы расстояние до них хотя бы на единицу превосходило расстояния до целевых соседей:

$$\sum_{i=1}^{\ell} \sum_{j \neq i} \sum_{\substack{m \neq i \\ m \neq j}} \eta_{ij} [y_m \neq y_i] \max(0, 1 + \|Ax_i - Ax_j\|^2 - \|Ax_i - Ax_m\|^2).$$

Суммируя эти два выражения, получим итоговый функционал:

$$\sum_{i \neq j} \eta_{ij} \|Ax_i - Ax_j\|^2 + \\ + C \sum_{i=1}^{\ell} \sum_{j \neq i} \sum_{\substack{m \neq i \\ m \neq j}} \eta_{ij} [y_m \neq y_i] \max(0, 1 + \|Ax_i - Ax_j\|^2 - \|Ax_i - Ax_m\|^2) \to \min_A$$

Данную задачу можно свести к стандартной задаче с линейным функционалом и ограничениями на неотрицательную определённость матрицы и решена стандартными солверами.

0.26 Опишите хотя бы два способа моделирования зависимостей между метками в multi-label классификации.

0.26.1 Multi-label классификации

В классической задаче многоклассовой классификации необходимо каждому объекту сопоставлять метку одного из классов, к которому этот объект принадлежит. На практике также встречаются задачи, где объект может принадлежать сразу нескольким классам. Такие задачи называют задачами классификации с пересекающимися классами (multi-label classification). Такие задачи встречаются, например, в задаче категорирования новостей, тегирования песен и изображений. Рассмотрим подходы к решению таких задач.

0.26.2 Multi-label задача как множество независимых задач бинарной классификации

В простом случае можно рассматривать каждую метку независимо, то есть решать задачу бинарной классификации "соответствует ли конкретный объект заданной метке". В таком случае необходимо обучить столько классификаторов, сколько меток существует в нашей задаче. Однако простота решения задачи как множество независимых задач оборачивается тем, что мы теряем связь между метками. Например, знание о том, что фильм является ужастиком, может сильно поменять предсказание "является ли фильм романтическим": $p(y_{\text{romance}}|x) \neq p(y_{\text{romance}}|x, y_{\text{horror}})$.

Упрощение модели можно представить аналогично наивному байесовскому классификатору (только факторизуем распределение не по признакам, а по меткам):

$$p(y_1, y_2, \ldots, y_D|x) \approx \prod_{i=1}^{D} p(y_i|x)$$

Тогда предсказание каждой метки:

$$\hat{y_i} = argmax_{y_i} p(y_i|x)$$

0.26.3 Моделирование зависимостей между метками

Используя определение условной вероятности, можно получить следующее соотношение:

$$p(y_1, y_2, \ldots, y_D|x) = \prod_{i=1}^{D} p(y_i|x, y_1, y_2, \ldots, y_{i-1})$$

Оно соответствует модели на рис. (1).

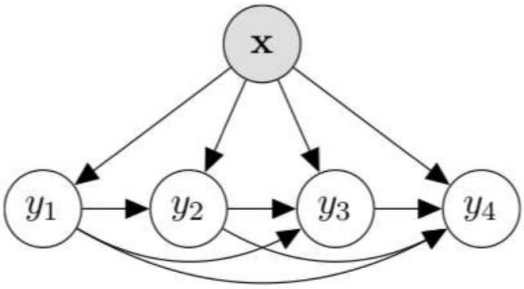


Рис. 1: Моделирование зависимостей между метками

При таком подходе предсказания нельзя получить независимо по меткам как в предыдущем случае. Существует несколько подходов:

1. Жадный подход заключается в последовательном применении $argmax_{y_i}$ в порядке всех меток. Решение может быть далёким от оптимального, но требует минимального числа вычислений.

2. Оптимальный подход заключается в расчёте всех вероятностей $p(y_1, y_2, \ldots, y_D|x)$ с последующим применением $argmax$ к совместном распределению (получится дерево всех возможных значений меток, где в листьях будут вероятности возможных $\hat{y}$). Число вершин будет расти экспоненционально, поэтому такой способ применим только в задачах с малым количеством меток (≈ 15).

3. Beam-search заключается в обогащении жадного подхода поддержкой не одного значения — текущего $argmax$, а сразу нескольких на случай, если путь с максимальной вероятностью на конкретном шаге не будет оптимальным в итоге.

4. Монте-Карло подход заключается в сэмплировании, по какому пути дерева пойти дальше.

С точки зрения обучения моделей и матрицы признаков это будет выглядеть так, что алгоритм получает для обучения матрицу признаков и метки или предсказания всех предшествующих меток. В таком случае нужно быть аккуратным и не допустить переобучения, подавая ответы на предшествующих метках следующим алгоритмам в качестве признаков (как это случается в стэкинге).

Сложностью этого подхода является то, что необходимо выбрать порядок меток, так как на практике мы делаем оценки на эти распрееления и формальное равенство распределений для любого порядка меток будет нарушаться. Также иногда можно исключить некоторые метки из условных распределений с правой стороны, если есть информация о том, что они не влияют на распределение (особенно, если в задаче много меток).

0.27 Метрики качества ранжирования: MAP, nDCG, pFound

MAP (Mean Average Precision)

Рассмотрим случай задачи ранжирования, когда таргет для каждого объекта $(q_i, d_i) \in X$ является $y_i \in \{0, 1\}$, т.е. релевантен ли документ d_i для запроса q_i. В таком случае введем следующие метрики:

- precision@$k(q)$ – точность, вычисленную для документов, которые модель поместила на первые k мест для запроса q

- AP@$k(q) = \sum_{i=1}^{k} \frac{y_{(i)}}{\sum_{j=1}^{k} y_{(j)}}$precision@$i(q)$, где $y_{(i)}$ – релевантность документа на i позиции.

- MAP@$k = \sum_q \text{AP@}k(q)$, т.е. усредняем Average Precision по всем запросам.

nDCG (normalized Discounted Cumulative Gain)

В случае если ответы вещественные, используют DCG

$$\text{DCG@k}(q) = \sum_{i=1}^{k} g(y_{(i)})d(i).$$

Примерами конкретных функций могут служить $g(y) = 2^y - 1$ и $d(i) = \frac{1}{\log(i+1)}$.

Для удобства интерпретации ее нормируют

$$\text{nDCG@k}(q) = \frac{\text{DCG@k}(q)}{\max \text{DCG@k}(q)}.$$

Далее она усредняется по всем запросам.

pFound

Пусть ответы лежат на отрезке $[0, 1]$ и отражают вероятность найти ответ в данном документе. Зададим величину p_i, соответствующую вероятности дойти до i-й позиции. Для первой позиции она равна единице, поскольку пользователь точно посмотрит на первый документ: $p_1 = 1$. Вероятность дойти до $(i + 1)$-й позиции вычисляется как

$$p_{i+1} = p_i(1 - y_{(i)})(1 - p_{\text{out}}),$$

где p_{out} — вероятность того, что пользователь уйдёт, не узнав ответ на свой запрос. Метрика pFound равна вероятности найти ответ среди первых k документов:

$$\text{pFound@}k(q) = \sum_{i=1}^{k} p_i y_{(i)}.$$

Далее она, как и другие метрики, усредняется по всем запросам.

0.28 В чём идея pointwise и pairwise подходов?

Pointwise

В поточечном (pointwise) подходе предлагается забыть про то, что мы решаем задачу ранжирования, и независимо для каждого объекта x_i предсказывать ответ y_i. В зависимости от типа ответов получим задачу классификации или регрессии.

Pairwise

Вспомним, что изначально мы определяли задачу ранжирования через пары объектов. Если записывать это формально, то получим функционал ошибки

$$\sum_{(i,j)\in R} [a(x_j) - a(x_i) < 0],$$

где R — множество пар, для которых известен порядок. Этот функционал не является дифференцируемым, но мы уже решали такую проблему в линейной классификации — надо заменить индикатор ошибки $[z < 0]$ на его гладкую верхнюю оценку $L(z)$:

$$\sum_{(i,j)\in R} [a(x_j) - a(x_i) < 0] \leq \sum_{(i,j)\in R} L\left(a(x_j) - a(x_i)\right).$$

В качестве оценки $L(z)$ можно брать, например, логистическую функцию $L(x) = \log(1+e^{-\sigma z})$ с параметром $\sigma > 0$ — в этом случае получим метод RankNet.

0.29 Общая схема метода LambdaRank.

Вспомним, что изначально мы определяли задачу ранжирования через пары объектов. Если записывать это формально, то получим функционал ошибки

$$\sum_{(i,j)\in R} [a(x_j) - a(x_i) < 0],$$

где R — множество пар, для которых известен порядок. Этот функционал не является дифференцируемым, но мы уже решали такую проблему в линейной классификации — надо заменить индикатор ошибки $[z < 0]$ на его гладкую верхнюю оценку $L(z)$:

$$\sum_{(i,j)\in R} [a(x_j) - a(x_i) < 0] \leq \sum_{(i,j)\in R} L\left(a(x_j) - a(x_i)\right).$$

В качестве оценки $L(z)$ можно брать, например, логистическую функцию $L(x) = \log(1+e^{-\sigma z})$ с параметром $\sigma > 0$ — в этом случае получим метод RankNet. Оптимизировать данный функционал можно обычным стохастическим градиентным спуском. Если использовать линейную модель $a(x) = \langle w, x\rangle$, то один шаг будет иметь вид

$$w := w + \eta \frac{\sigma}{1 + \exp(\sigma\langle x_j - x_i, w\rangle)}(x_j - x_i).$$

Существует эмпирическое наблюдение, позволяющее перейти к оптимизации произвольной метрики ранжирования F. Оказывается, для этого надо домножить смещение на изменение метрики ΔF_{ij}, которое произойдёт при перестановке x_i и x_j местами в ранжировании:

$$w := w + \eta \frac{\sigma}{1 + \exp(\sigma\langle x_j - x_i, w\rangle)}|\Delta F_{ij}|(x_j - x_i).$$

Данный метод носит название **LambdaRank**.

0.30 Опишите метод ListNet.

Мы уже отмечали выше, что непосредственно оптимизировать метрику качества ранжирования вряд ли получится из-за её дискретной структуры. Будем решать данную проблему путём введения некоторого вероятностного распределения. Метод **ListNet** отличается от других методов тем, что позволяет непосредственно учитывать порядок объектов в процессе обучения.

Рассмотрим один запрос q, для которого надо отранжировать n_q документов $(d_1, \ldots, d_{n_q})$. Для этих документов известны истинные оценки релевантности $(y_1, \ldots, y_{n_q})$, которые определяют истинное ранжирование. Пусть также имеет некоторая модель $a(x)$, которая выдаёт оценки $(z_1, \ldots, z_{n_q})$. Метрика качества ранжирования (например, nDCG) измеряет, насколько ранжирования по оценкам модели близко к истинному ранжированию.

В постановке, которую мы только что описали, модель выдаёт одну конкретную перестановку документов для данного запроса, и мы измеряем качество этой перестановки. Сгладим этот процесс: предположим, что на самом деле модель выдаёт распределение на всех возможных перестановках документов, причём вероятность конкретной перестановки π определяется как

$$P_z(\pi) = \prod_{j=1}^{n_q} \frac{\phi(z_{\pi(j)})}{\sum_{k=j}^{n_q} \phi(z_{\pi(k)})},$$

где ϕ — произвольная неубывающая и строго положительная функция. Данные вероятности обладают рядом полезных свойств:

- Вероятности $P_z(\pi)$ задают распределение на множестве всех перестановок n_q элементов.
- Пусть перестановка π ставит объект x_i выше объекта x_j, и $z_i > z_j$. Если поменять эти объекты местами в перестановке (то есть поставить x_j выше, чем x_i), то новая перестановка будет иметь меньшую вероятность, чем старая. Иными словами, чем ближе перестановка к оптимальной, тем выше её вероятность.

- Максимальную вероятность имеет перестановка, которая сортирует объекты по убыванию z_i; минимальную вероятность имеет обратная к ней перестановка.

Таким образом, данные вероятности отдают предпочтение тем перестановкам, которые ближе к сортировке объектов по предсказаниям алгоритма. Значит, мы действительно получим способ «сглаживания» ранжирования документов по прогнозам z_i.

Всего перестановок объектов $n_q!$, и посчитать по ним всем матожидание не представляется возможным. Чтобы упростить подсчёты, рассмотрим вероятность $P_z(j)$ попадания объекта x_j на первое место после перестановки. Можно показать, что они вычисляются по формуле

$$P_z(j) = \frac{\phi(z_j)}{\sum_{k=1}^{n} \phi(z_k)}$$

Данные вероятности образуют распределение на всех n_q документах. Также для объектов с $z_i > z_j$ выполнено $P_z(i) > P_z(j)$ — то есть введённые вероятности задают на объектах порядок, совпадающий с ранжированием по оценкам модели.

Теперь мы можем сравнить истинное ранжирование документов и ранжирование по оценкам модели, посчитав кросс-энтропию между соответствующими им распределениями:

$$Q(y, z) = -\sum_{j=1}^{n_q} P_y(j) \log P_z(j).$$

Если взять, например, $\phi(x) = \exp(x)$, то кросс-энтропия будет дифференцируема по оценкам модели — а значит, можно найти производные и по параметрам модели. Благодаря сглаживанию мы смогли ввести функционал, который отражает требования к перестановке объектов, и при этом позволяет обучение градиентными методами.

0.31 Опишите метод SoftRank.

Вместо точечной оценки score для конкретного запроса и j-го документа будем считать, что нам выдаётся распределение (байесовский подход):

$$p(s_j) = \mathcal{N}(a(q, d_j), \sigma_s^2)$$

Посчитаем вероятность того, что i-ый документ окажется выше j-го:

$$\pi_{ij} = P(s_i > s_j) = P(\underbrace{s_i - s_j}_{\mathcal{N}(a(q,d_i)-a(q,d_j),2\sigma^2)} > 0) = \int_0^\infty \mathcal{N}(a(q, d_i) - a(q, d_j), 2\sigma^2) ds$$

Хотим найти r_j - позиция, на которую встаёт d_j в ранжировании по модели (поскольку в соревновании с другим документом, он побеждает с какой-то вероятностью, то тогда его позиция тоже будет случайной величиной).

Данное распределение называется **Rank-Binomial distribution** (кол-во успехов в $N-1$ соревновании, у каждого из которых своя вероятность). Вероятности вычисляются итерационно:

$$p_j^{(1)}(r) = [r = 0]$$
$$p_j^{(i)}(r) = p_j^{(i-1)}(r-1)\pi_{ij} + p_j^{(i-1)}(r)(1 - \pi_{ij})$$

Пояснение - j-ый документ может попасть на позицию r двумя способами:

$p_j^{(i-1)}(r-1)\pi_{ij}$ – документ стоял на $r-1$ позиции и проиграл i-ому документу, т.е. сместился на позицию r

$p_j^{(i-1)}(r)(1-\pi_{ij})$ – документ уже стоял на позиции r и победил i-ый документ, т.е. остался на позиции r

0.32 Blade-chest модель.

Постановка задачи

Предположим, что мы хотим решать задачу ранжирования, но наши данные представлены лишь попарными взаимодействиями между объектами, то есть для некоторого набора объектов $\{x_i\}_{i=1}^n$ обучающая выборка представлена случайным подмножеством попарных соотношений: $\{x_{i_k} > x_{j_k}\}_{k=1}^\ell$. Также для таких соотношений может быть известен некий контекст. Далее мы будем рассматривать модели для попарных соотношений в контексте матчей между двумя игроками x_i, x_j, при этом наши рассуждения могут быть полностью перенесены на все остальные случаи применения этих моделей.

В общем случае данные о попарных соотношениях выглядят следующим образом:

- признаки объектов - для каждого игрока известно его признаковое описание $x_i \in \mathbb{R}^d$ (например, id, вес, рост, национальность и т.п.);
- признаки контекста - для каждого матча известно его признаковое описание $z_k \in \mathbb{R}^D$ (например, погода, время, место и т.п.);
- набор соотношений - история матчей в виде троек $\{(x_{i_k}, x_{j_k}, z_k)\}$, где на первом месте стоит победитель матча.

Пример для понимания

Допустим, мы хотим рекомендовать рестораны пользователям какой-нибудь Яндекс.Еды. Тогда объекты – это рестораны. Признаки контекста – это признаки пользователя, которому мы хотим ресторан порекомендовать. Отношение – это событие, когда пользователь предпочел один ресторан другому.

То есть если пользователь выбирал между «Додо пиццей» и «Domino's» и в итоге заказал пиццу из «Domino's», то в обучающую выборку добавляется тройка («Domino's», «Додо пицца», признаки пользователя).

Формулировка blade-chest модели

Для каждого объекта введём два вектора: x_i^{blade} и x_i^{chest}. Чтобы оценить вероятность победы игрока x_i над игроком x_j, нужно сравнить эти векторы - если x_i^{blade} ближе к x_j^{chest}, чем x_j^{blade} к x_i^{chest}, то вероятнее победит первый игрок и наоборот.

Введём функцию $M(x_i, x_j)$ следующим образом:

$$M(x_i, x_j) = \|x_j^{blade} - x_i^{chest}\|^2 - \|x_i^{blade} - x_j^{chest}\|^2$$

Можно использовать альтернативный вариант:

$$M(x_i, x_j) = \langle x_i^{blade}, x_j^{chest} \rangle - \langle x_j^{blade}, x_i^{chest} \rangle$$

Наша модель оценивает исход матча вот таким простым образом:

$$P(x_i \text{ wins } x_j) = \sigma\left(M(x_i, x_j)\right)$$

Откуда брать x_i^{blade} и x_i^{chest}?

Для оценки этих векторов можно использовать линейную комбинацию признаков игроков и любую дифференцируемую функцию:

$$\begin{aligned} x_i^{blade} &= f_{blade}(Bx_i) \\ x_j^{blade} &= f_{blade}(Bx_j) \\ x_i^{chest} &= f_{chest}(Cx_i) \\ x_j^{chest} &= f_{chest}(Cx_i) \end{aligned}$$

Формулы выше можно рассматривать как однослойный перцептрон с активацией. Без ограничений общности тут можно использовать любую нейронку.

Как учитывать признаки контекста z_k?

1. Первый способ заключается в добавлении признаков игры к признаковому описанию каждого игрока:

$$x_i^{\text{blade}} = f_{\text{blade}}\left(B\begin{bmatrix} x_i \\ z_k \end{bmatrix}\right),$$
$$x_i^{\text{chest}} = f_{\text{chest}}\left(C\begin{bmatrix} x_i \\ z_k \end{bmatrix}\right).$$

2. Второй способ заключается в поэлементном домножении векторов представлений на векторы представлений матча, полученные из признакового описания матча аналогично векторам $x^{\text{blade}}, x^{\text{chest}}$, то есть:

$$x_i^{\text{blade}} = f_{\text{blade}}\ (Bx_i) \cdot f_{\text{match}}\ (B'z_k),$$
$$x_i^{\text{chest}} = f_{\text{chest}}\ (Cx_i) \cdot f_{\text{match}}\ (C'z_k)$$

Итак, как предсказывать?

1. Нам на вход приходят x_i, x_j, z.
2. По формулам выше находим x_i^{blade}, x_i^{chest}, x_j^{blade} и x_j^{chest}.
3. На основе этих векторов вычисляем $M(x_i, x_j)$.
4. $P(x_i \text{ wins } x_j) = \sigma\left(M(x_i, x_j)\right)$.

Ну ладно, так как это всё учить?

Как мы помним, обучающая выборка состоит из троек $\{(x_{i_k}, x_{j_k}, z_k)\}$, причём известно, что ($x_{i_k}$ победил x_{j_k}. Заметим, что в нашей модели $M(x_i, x_j)$ – дифференцируемая функция от x_{i_k}, x_{j_k}, z_k. Тогда мы можем напрямую оптимизировать лог-правдоподобие модели.

$$\sum_{k=1}^{\ell} \log P\left(x_{i_k} \text{ wins } x_{j_k}\right) \to \max$$

0.33 User-based и item-based подходы к рекомендациям.

Постановка задач в рекомендательных системах

Мы будем рассуждать в терминах пользователей (users, U) и товаров (items, I), но описанные далее методы подходят для рекомендаций любых объектов. Будем считать, что для некоторых пар пользователей $u \in U$ и товаров $i \in I$ известны оценки r_{ui}, которые отражают степень заинтересованности пользователя в товаре.

Требуется по известным рейтингам r_{ui} научиться строить для каждого пользователя u набор из k товаров $I(u)$, наиболее подходящих данному пользователю - то есть таких, для которых рейтинг r_{ui} окажется максимальным.

Получается понятная задача: для объекта x_{ui} «пользователь-товар» нужно предсказать значение целевой переменной r_{ui}.

User-based collaborative filtering

Два пользователя похожи, если они ставят товарам одинаковые оценки. Рассмотрим двух пользователей u и v. Обозначим через I_{uv} множество товаров i, для которых известны оценки обоих пользователей:

$$I_{uv} = \{i \in I \mid \exists r_{ui}\ \&\ \exists r_{vi}\}$$

Тогда сходство двух данных пользователей можно вычислить через корреляцию Пирсона между оценками, которые они ставят товарам:

$$w_{uv} = \frac{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \bar{r}_v)^2}},$$

где $\bar{r}_u$ и $\bar{r}_v$ — средние рейтинги пользователей по множеству товаров I_{uv}.

В подходе на основе сходств пользователей (user-based collaborative filtering) определяется множество $U(u_0)$ пользователей, похожих на данного:

$$U(u_0) = \{v \in U \mid w_{u_0 v} > \alpha\}.$$

После этого для каждого товара вычисляется, как часто он покупался пользователями из $U(u_0)$:

$$p_i = \frac{|\{u \in U(u_0) \mid \exists r_{ui}\}|}{|U(u_0)|}.$$

Пользователю рекомендуются k товаров с наибольшими значениями p_i. Данный подход позволяет строить рекомендации, если для данного пользователя найдутся похожие. Если же пользователь является нетипичным, то подобрать что-либо не получится.

Item-based collaborative filtering

Здесь всё симметрично user-based collaborative filtering.

Чтобы вычислять сходства между товарами i и j, введём множество пользователей U_{ij}, для которых известны рейтинги этих товаров:

$$U_{ij} = \{u \in U \mid \exists r_{ui} \;\&\; \exists r_{uj}\}.$$

Тогда сходство двух данных товаров можно вычислить через корреляцию Пирсона:

$$w_{ij} = \frac{\sum_{u \in U_{ij}} (r_{ui} - \bar{r}_i)(r_{uj} - \bar{r}_j)}{\sqrt{\sum_{u \in U_{ij}} (r_{ui} - \bar{r}_i)^2} \sqrt{\sum_{u \in U_{ij}} (r_{uj} - \bar{r}_j)^2}},$$

где $\bar{r}_i$ и $\bar{r}_j$ — средние рейтинги товаров по множеству пользователей U_{ij}. Отметим, что существуют и другие способы вычисления похожестей - например, можно вычислять скалярные произведения между векторами рейтингов двух товаров.

В подходе item-based collaborative filtering определяется множество товаров, похожих на те, которые интересовали данного пользователя:

$$I(u_0) = \{i \in I \mid \exists r_{u_0 i_0}, w_{i_0 i} > \alpha\}.$$

Затем для каждого товара из этого множества вычисляется его сходство с пользователем:

$$p_i = \max_{i_0 : \exists r_{u_0 i_0}} w_{i_0 i}.$$

Пользователю рекомендуются k товаров с наибольшими значениями p_i. Даже если пользователь нетипичный, то данный подход может найти товары, похожие на интересные ему — и для этого необязательно иметь пользователя со схожими интересами.

0.34 Как выглядит модель со скрытыми переменными для рекомендательных систем? Какие данные необходимы для её обучения? Какие методы обучения этой модели вы знаете?

Модель со скрытыми переменными (Latent Factor Model)

Будем строить для каждого юзера u и айтема i векторы $p_u, q_i \in \mathbb{R}^d$, которые будут эмбедингами "категорий интересов". По идее, компоненты юзера можно интерпретировать как степень заинтересованности в категории, а

компоненты айтема как степень принадлежности к категории. Впрочем, никто не гарантирует, что эти компоненты будут соответствовать каким-то реальным категориям.

Сходство юзера и айтема будем оценивать скалярным произведением эмбедингов:

$$r_{ui} \approx \langle p_u, q_i \rangle + \overline{r}_i + \overline{r}_u$$

То есть мы считаем, что r_{ui} раскладывается на средний рейтинг айтема, средний рейтинг пользователя и некое смещение, которое мы оцениваем скалярным произведением. *Важно:* в конспектах лекции прошлого года формула приводится без $\overline{r}_i$ и $\overline{r}_u$, но мы на семинаре решили, что так не должно быть и это ошибка.

Также скалярным произведением можно оценить сходство двух юзеров или двух айтемов.

Для данной задачи мы можем сформулировать следующий функционал ошибки:

$$\sum_{(u,i)\in R} \left(r_{ui} - \overline{r}_u - \overline{r}_i - \langle p_u, q_i \rangle\right)^2 \to \min_{P,Q}$$

Здесь мы суммируем по всем парам (u, i), для которых известен рейтинг. Заметим, что если матрица R' - R с центрированными строками и столбцами, то задача сводится к низкоранговому матричному разложению:

$$||R' - P^T Q||_2^2 \to \min_{P,Q}$$

Здесь представления юзеров и айтемов записаны в столбцах матриц P и Q.

Можно домножать скалярные произведения на масштабирующий множитель α:

$$||R' - \alpha P^T Q||_2^2 \to \min_{\alpha,P,Q}$$

Можно регуляризировать эмбединги:

$$\sum_{(u,i)\in R} \left(r_{ui} - \overline{r}_u - \overline{r}_i - \langle p_u, q_i \rangle\right)^2 + \lambda \sum_u ||p_u||^2 + \mu \sum_i ||q_i||^2 \to \min_{P,Q}$$

Собственно, это и есть LFM.

Какие данные необходимы для обучения LFM?

Да все те же самые, что и для обучения любых других методов рекомендаций. У нас есть множество пользователей U и множество товаров I, а также разреженная матрица $R \in \mathbb{R}^{|U|\times|I|}$, где r_{ui} – оценка, которую пользователь u поставил товару i, если между ними было взаимодействие. Под оценкой здесь необязательно понимается явный рейтинг, а вообще любой сигнал: прослушивание трека до конца, покупка товара, лайк/дизлайк и так далее.

Какие методы обучения LFM вы знаете?

Существует два основных подхода к оптимизации функционала LFM. Первый - стохастический градиентный спуск, который на каждом шаге случайно выбирает пару $(u, i) \in R$:

$$p_{uk} := p_{uk} + \eta q_{ik} \left(r_{ui} - \bar{r}_u - \bar{r}_i - \langle p_u, q_i \rangle\right)$$
$$q_{ik} := q_{ik} + \eta p_{uk} \left(r_{ui} - \bar{r}_u - \bar{r}_i - \langle p_u, q_i \rangle\right)$$

Второй подход основан на особенностях функционала и называется ALS (alternating least squares). Можно показать, что этот функционал не является выпуклым в совокупности по P и Q, но при это становится выпуклым, если зафиксировать либо P, либо Q. Более того, оптимальное значение P при фиксированном Q (и наоборот) можно выписать аналитически, - но оно будет содержать обращение матрицы:

$$p_u = \left(\sum_{i:\exists r_{ui}} q_i q_i^T\right)^{-1} \sum_{i:\exists r_{ui}} r_{ui} q_i$$
$$q_i = \left(\sum_{u:\exists r_{ui}} p_u p_u^T\right)^{-1} \sum_{u:\exists r_{ui}} r_{ui} p_u$$

(здесь через p_u и q_i мы обозначили столбцы матриц P и Q).

Чтобы избежать сложной операции обращения, будем фиксировать всё, кроме одной строки p_k матрицы P или одной строки q_k матрицы Q. В этом случае можно найти оптимальное значение для p_k и q_k :

$$p_k = \frac{q_k \left(R - \sum_{s \neq k} p_s q_s^T\right)^T}{q_k q_k^T},$$
$$q_k = \frac{p_k \left(R - \sum_{s \neq k} p_s q_s^T\right)}{p_k p_k^T}.$$

Данный подход носит название Hierarchical alternating least squares (HALS).

Implicit ALS

Я не вполне уверен, что этот материал входит в вопрос, но на всякий случай добавлю его сюда.

Все подходы выше полагаются на **явный** (explicit) фидбэк от пользователя, когда он каким-то образом может сообщить, что товар ему понравился (поставил лайк или 5 звёзд) или не понравился (поставил дизлайк или 1 звезду). Но часто фидбэк у нас есть только **неявный** (implicit), когда мы знаем лишь, что пользователь как-то взаимодействовал с товаром, но понравилось ли ему – неизвестно. Например, пользователь посмотрел фильм, но не поставил ему оценку: в таком случае мы можем только гадать, нужно нам рекомендовать похожие фильмы или не нужно.

К счастью, в семействе ALS есть герой, которого мы заслуживаем и который нужен в таких ситуациях: Implicit ALS (iALS)!

Введём показатель неявного интереса пользователя к товару:

$$s_{ui} = \begin{cases} 1, & \exists r_{ui}, \\ 0, & \text{иначе.} \end{cases}$$

Также введем веса c_{ui}, характеризующие уверенность в показателе интереса s_{ui} :

$$c_{ui} = 1 + \alpha r_{ui}.$$

Коэффициент α позволяет регулировать влияние явного рейтинга на уверенность в интересе. Теперь мы можем задать функционал:

$$\sum_{(u,i) \in D} c_{ui} \left(s_{ui} - \bar{s}_u - \bar{s}_i - \langle p_u, q_i \rangle\right)^2 + \lambda \sum_u \|p_u\|^2 + \mu \sum_i \|q_i\|^2 \to \min_{P,Q}$$

Как и раньше, обучать его можно с помощью стохастического градиентного спуска, ALS или HALS. Предложенные способы вычисления s_{ui} и c_{ui} могут изменяться в зависимости от специфики задачи.

0.35 Опишите модель factorization machine.

Factorization machine – это такая чудесная штука, которую можно представить через регрессию и через матричные разложения.

FM как полиномиальная регрессия

Рассмотрим признаковое пространство $\mathbb{R}^d$. Допустим, что целевая переменная зависит от парных взаимодействий между признаками. В этом случае представляется разумным строить полиномиальную регрессию второго порядка:

$$a(x) = w_0 + \sum_{j=1}^{d} w_j x_j + \sum_{j_1=1}^{d} \sum_{j_2=j_1+1}^{d} w_{j_1 j_2} x_{j_1} x_{j_2}.$$

Данная модель состоит из $d(d-1)/2+d+1$ параметров. Если среди признаков есть категориальные с большим числом категорий (например, идентификатор пользователя), то после их бинарного кодирования число параметров станет слишком большим. Чтобы решить проблему, предположим, что вес взаимодействия признаков j_1 и j_2 может быть аппроксимирован произведением низкоразмерных скрытых векторов v_{j_1} и v_{j_2}, характеризующих эти признаки. Мы получим модель, называемую факторизационной машиной (factorization machine, FM):

$$a(x) = w_0 + \sum_{j=1}^{d} w_j x_j + \sum_{j_1=1}^{d} \sum_{j_2=j_1+1}^{d} \langle v_{j_1}, v_{j_2} \rangle x_{j_1} x_{j_2}.$$

Благодаря описанному трюку число параметров снижается до $dr + d + 1$, где $r-$ размерность скрытых векторов.

FM как матричное разложение

Данная модель является обобщением моделей с матричными разложениями. Задачу предсказания рейтинга по товару и пользователю в целом можно сформулировать как задачу построения регрессии с двумя категориальными признаками: идентификатором пользователя и идентификатором товара. Целевым признаком является рейтинг r_{ui}. Для некоторого подмножества пар (пользователь, товар) мы знаем рейтинг; для остальных мы хотим его восстановить. После бинаризации признаков получим, что каждый объект x описывается $|U| + |I|$ признаками, причём ненулевыми являются ровно два из них: один соответствует номеру пользователя u, второй - номеру товара i. Тогда факторизационная машина примет следующий вид:

$$a(x) = w_0 + w_u + w_i + \langle v_u, v_i \rangle$$

Данная форма полностью соответствует модели. По сути, факторизационная машина позволяет строить рекомендательные модели на основе большого количества категориальных и вещественных признаков.

Существует несколько методов настройки факторизационных машин, из которых наиболее совершенным считается метод Монте-Карло на основе марковских цепей; реализацию можно найти в библиотеке libFM.

Field-aware Factorization Machine

Недавно было предложено расширение факторизационных машин, позволившее авторам победить в конкурсах Criteo и Avazu по предсказанию кликов по рекламным объявлениям. В обычных факторизационных машинах у каждого признака имеется всего один скрытый вектор, отвечающий за взаимодействие с остальными признаками. Допустим, что признаки можно некоторым образом сгруппировать например, в задаче рекомендации музыкальных альбомов в бинарном векторе, отвечающем за композиции, будет стоять несколько единиц, соответствующих всем композициям из альбома. Все единицы из этого вектора можно объединить в одну группу. Расширим модель, введя для каждого признака разные скрытые векторы для взаимодействия с разными группами:

$$a(x) = w_0 + \sum_{j=1}^{d} w_j x_j + \sum_{j_1=1}^{d} \sum_{j_2=j_1+1}^{d} \left\langle v_{j_1,f_{j_2}}, v_{j_2,f_{j_1}} \right\rangle x_{j_1} x_{j_2}$$

где f_{j_1} и f_{j_2} - индексы групп признаков x_{j_1} и x_{j_2}. Данная модель носит название field-aware factorization machines (FFM).

0.36 Опишите метод LIME для интерпретации моделей.

Рассмотрим метод **LIME** (*Local Interpretable Model-Agnostic Explanations*), основная идея которого, как следует из названия, заключается в том, чтобы интерпретировать предсказания некоторой **объясняемой** модели $a(x)$ для заданного объекта x^* в его окрестности.

Предполагается, что полученная интерпретация может использоваться широким кругом лиц для принятия решений (например, докторами для принятия решений о лечении пациентов), поэтому для каждого объекта наряду

с признаковым описанием x, используемым в модели $a(x)$, вводится его *интерпретируемое* представление $\bar{x}$, а результатом работы метода является **объясняющая** модель $\bar{a}(\bar{x})$, строящая предсказания именно для этих интерпретируемых представлений (а не для исходных признаковых описаний объекта). В связи с этим в качестве интерпретируемых представлений обычно рассматривают достаточно простые бинарные признаковые описания исходных объектов, например:

- для текстовых данных можно использовать «мешок слов» с ограничением на количество слов или N-грамм, используемых в представлении;
- для изображений используется аналогичное представление, но вместо слов выступают *суперпиксели* — непрерывные области «похожих» пикселей, которые могут быть найдены на изображении x при помощи любого стандартного метода сегментации изображений.

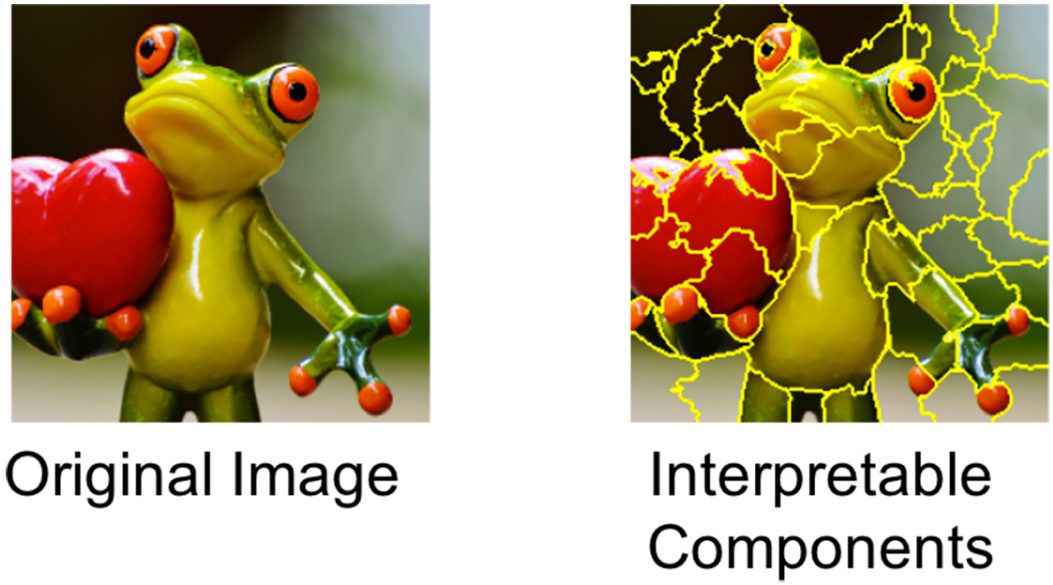


Рис. 2: Пример разбиения изображения на суперпиксели

Для построения объясняющей модели $\bar{a}(\bar{x})$ составляется «суррогатная» выборка $X^{\ell}_{x^*} = \{(\bar{x}_i, y_i)\}$ следующим образом: создадим объект $\bar{x}_i$ путем случайного обнуления случайного количества единиц (все случайности — согласно равномерным распределениям) в интерпретируемом представлении $\bar{x}^*$ объекта, для которого строится интерпретация, после чего перейдем от представления $\bar{x}_i$ нового объекта к признаковому описанию x_i в исходном признаковом пространстве и положим $y_i = a(x_i)$. Таким образом, мы создали искусственную выборку в окрестности интерпретируемого представления $\bar{x}^*$, после чего для каждого объекта вычислили таргет как прогноз интерпретируемой модели $a(x)$ в исходном признаковом пространстве (в случае многоклассовой классификации интерпретация чаще всего строится независимо для каждого класса).

Обратим внимание на переход от интерпретируемого представления $\bar{x}_i$ к исходному признаковому описанию x_i: для текстов под «обнулением» элементов представления понимается удаление соответствующих слов или N-грамм из исходного текста и вычисление нового признакового описания для изменённого текста; для изображений аналогично подразумевается «закрытие» соответствующих суперпикселей изображения.

Поскольку итоговая модель $\bar{a}(\bar{x})$ должна, во-первых, интерпретировать предсказания в окрестности исходного объекта, и, во-вторых, быть достаточно простой, то она может быть найдена как решение, например, следующей

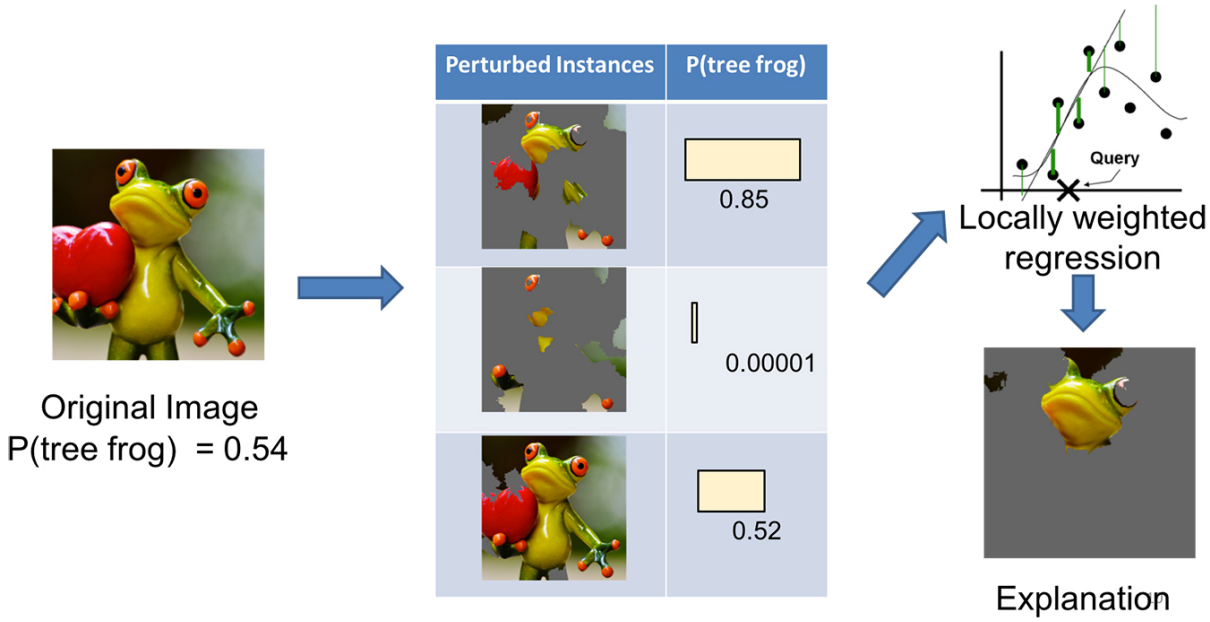


Рис. 3: Пример генерации объектов для суррогатной выборки: из исходного изображения случайным образом удаляется часть суперпикселей, для полученного изображения вычисляется прогноз модели, после чего на итоговой взвешенной выборке интерпретируемых бинарных представлений обучается объясняющая модель

задачи:

$$\bar{a} = argmin_{b \in B} \sum_{\bar{x}_i \in X^{\ell}_{x^*}} \pi_{x^*}(x_i)(a(x_i) - b(\bar{x}_i))^2 + \Omega(b),$$

где x^* — объект, для которого происходит поиск интерпретации,

B — семейство возможных объясняющих моделей (ещё раз обратим внимание на то, что эти модели строят предсказания для интерпретируемых представлений, а не для исходных признаковых описаний),

$\pi_{x^*}(x_i)$ — вес объекта в функционале ошибки (часто в качестве $\pi_{x^*}(x)$ выбирают некоторое ядро в центром в объекте x^*),

$\Omega(b)$ — сложность объясняющей модели.

В качестве семейства объясняющих моделей B можно выбирать любое семейство достаточно простых моделей (например, линейные модели, решающие деревья или решающие списки), в качестве функции потерь можно также выбирать любую другую вместо квадратичной, используемой в формуле выше. Частный случай LIME при использовании квадратичной функции потерь с экспоненциальным ядром в качестве функции $\pi_{x^*}(x)$, а также семейства B линейных моделей и $\Omega(b) = \infty[\|w_b\|_0 > K]$, где w_b — вектор весов для модели b (то есть с ограничением количества ненулевых весов модели), называется методом *разреженных линейных представлений* (*Sparse Linear Explanations*) и на практике используется чаще всего.

После нахождения объясняющей модели $\bar{a}(\bar{x})$ в силу того, что она строит прогнозы для бинарных векторов, а также того, что является достаточно простой в силу выбора семейства и наличия регуляризации $\Omega(b)$, она может быть легко интерпретирована — например, в случае метода Sparse Linear Explanations в качестве интерпретации достаточно предъявить список признаков с ненулевыми весами (в случае текстов и изображений эти признаки говорят о наличии слов/N-грамм и суперпикселей соответственно).

0.37 В чём заключается идея Influence Function?

Недостатком подхода с удалением является его сложность — на практике обучить модели по количеству объектов в обучающей выборке не представляется возможным. Альтернативный вариант предлагает для моделей с диффе-

ренцируемой функцией потерь исследовать влияние изменения веса для конкретного примера обучающей выборки на параметры модели через влияние на функцию потерь.

Обозначим за $\hat{\theta}$ вектор параметров модели, обученной на выборке X^ℓ,
за $\hat{\theta}_{\varepsilon,x_i}$ — вектор параметров модели при увеличении веса объекта x_i на ε:

$$\hat{\theta}_{\varepsilon,x_i} = argmin_\theta \frac{1}{\ell}\sum_{i=1}^{\ell} L(x_i,\theta) + \varepsilon L(x_i,\theta)$$

Заметим, что $\hat{\theta}_{0,x_i} = \hat{\theta}$. Тогда влияние увеличения веса объекта x_i на ошибку на новом объекте x можно вычислить следующим образом:

$$I_{\text{up,loss}}(x_i,x) = \frac{\partial L(x,\hat{\theta}_{\varepsilon,x_i})}{\partial\varepsilon}\bigg|_{\varepsilon=0} = \nabla_\theta L(x,\hat{\theta}_{0,x_i})^\top \frac{d\hat{\theta}_{\varepsilon,x_i}}{d\varepsilon}\bigg|_{\varepsilon=0} =$$
$$= -\nabla_\theta L(x,\hat{\theta})^\top H_{\hat{\theta}}^{-1} \nabla_\theta L(x_i,\hat{\theta}), \tag{1}$$

где $H_{\hat{\theta}} = \frac{1}{\ell}\sum_1^\ell \nabla_\theta^2 L(x_i,\hat{\theta})$ — гессиан функции потерь L, который может быть вычислен приближённо. Кроме того, в последнем переходе мы воспользовались классическим результатом теории функций влияния о влиянии веса объекта на параметры модели, полное доказательство которого можно найти в **?**:

$$\frac{d\hat{\theta}_{\varepsilon,x_i}}{d\varepsilon}\bigg|_{\varepsilon=0} = -H_{\hat{\theta}}^{-1}\nabla_\theta L(x_i,\hat{\theta}).$$

Интуиция за этой формулой 1 следующая. Представим гессиан равным единичной матрице, тогда положительное значение $I_{\text{up,loss}}(x_i,x)$ (т.е. ухудшение качества предсказания на объекте x) означает противоположные направления градиентов функции потерь для объектов x_i и x (то есть объект x_i «мешает» построению хорошего прогноза на x).

Использовать функции влияния можно несколькими способами:

1. Сравнивать модели между собой. Как в примере с изображением, может получиться так, что одна из моделей ищет более сложные паттерны на изображении по сравнению с другой. Для этого нужно сравнивать между собой наиболее влиятельные изображения для некоторого примера выборки.

2. Детектирование несовпадений доменов между обучающим и тестовым множеством. Можно найти ложное срабатывание модели и изучить влиятельные объекты для этого ложного предсказания. Так можно выяснить паттерны в данных, которые мешают корректной работе алгоритма на новом домене данных (данные из несколько другого распределения).

3. Коррекция обучающих данных. Если у нас есть возможность перепроверить корректность разметки небольшого числа объектов обучающей выборки, то эффективнее сделать это на наиболее влиятельных объектах, так как именно они влияют на нашу модель сильнее всего.

0.38 Опишите метод FGSM для атаки на модель с известными параметрами и функцией потерь.

Пусть x — исходное изображение, для которого производится атака, y_{true} — его истинная метка. Разберем несколько широко используемых состязательных white-box атак.

1. **Fast gradient sign method** (FGSM)
   Ненаправленный метод атак, идея которого заключается в том, чтобы изменить объект в сторону градиента функции потерь L для истинной метки объекта:

$$x^{adv} = x + \varepsilon sign(\nabla_x L(x, y_{\text{true}})).$$

Заметим, что при таком способе изменения исходного изображения l_∞-норма их разности не будет превышать ε, поэтому мощность атаки может регулироваться этим гиперпараметром. Данный метод является ненаправленным, а также относится к т.н. одношаговым (*one-shot*) методам.

2. **Targeted fast gradient sign method** (T-FGSM)
   Направленная версия *FGSM*, в которой объект изменяется в сторону антиградиента функции потерь для целевой метки атаки y_{target} :
   $$x^{adv} = x - \varepsilon sign(\nabla_x L(x, y_{\text{target}})).$$

3. **Iterative fast gradient sign method** (I-FGSM)
   В отличие от предыдущих методов, являющихся одношаговыми, в данном вместо одного шага длины ε делается T шагов длины $\alpha = \frac{\varepsilon}{T}$:
   $$x_0^{adv} = x,$$
   $$x_{t+1}^{adv} = x_t^{adv} + \alpha sign(\nabla_x L(x, y_{\text{true}}))$$

В случае с black-box атаками необходимо обучить суррогатную модель на выборке, таргетами в которой являются предсказания атакуемой модели («чёрного ящика»), после чего для полученной модели проводится атака white-box методами.

0.39 Задача 1. Предложите модификацию user-based коллаборативной фильтрации, которая будет работать на данных с неявной информацией.

Дисклеймер. Данное решение является лишь продуктом воображения техера, никем не проверялось на корректность и **может оказаться неверным!**

Что такое user-based коллаборативная фильтрация, было описано в билете 33. Идея в том, что мы для каждого пользователя находим множество пользователей, похожих на него, и смотрим, какие товары нравятся пользователям из этого множества.

Важно, что похожих пользователей мы находим при помощи корреляции Пирсона:

$$I_{uv} = \{i \in I \mid \exists r_{ui} \;\&\; \exists r_{vi}\}$$
$$w_{uv} = \frac{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)(r_{vi} - \bar{r}_v)}{\sqrt{\sum_{i \in I_{uv}} (r_{ui} - \bar{r}_u)^2} \sqrt{\sum_{i \in I_{uv}} (r_{vi} - \bar{r}_v)^2}}$$

Здесь w_{uv} – наша мера похожести пользователя u на пользователя v.

Теперь вспомним, что такое неявная информация. Об этом есть в билете 34 в части про Implicit ALS. Процитируем: «Все подходы выше полагаются на **явный** (explicit) фидбэк от пользователя, когда он каким-то образом может сообщить, что товар ему понравился (поставил лайк или 5 звёзд) или не понравился (поставил дизлайк или 1 звезду). Но часто фидбэк у нас есть только **неявный** (implicit), когда мы знаем лишь, что пользователь как-то взаимодействовал с товаром, но понравилось ли ему – неизвестно. Например, пользователь посмотрел фильм, но не поставил ему оценку: в таком случае мы можем только гадать, нужно нам рекомендовать похожие фильмы или не нужно».

Таким образом, в данных с явным фидбэком r_{ui} может принимать только два значения: 0 (если пользователь u с товаром i никак не взаимодействовал) и 1 (если какое-то взаимодействие было).

Становится довольно очевидно, что описанный выше способ вычисления похожести пользователей работать для таких данных не будет: для любых двух пользователей w_{uv} оказывается равным 1. Соответственно, нам нужно придумать какую-то другую формулу для w_{uv}, чтобы она учитывала особенности данных и w_{uv} было большим, если u и v взаимодействуют примерно с одними и теми же товарами, и маленьким – если u и v взаимодействуют с разными товарами.

Внимательный читатель уже догадывается, что в качестве этой формулы можно использовать меру Жаккара. Формально:

$$I_u = \{i \in I \mid \exists r_{ui}\}$$
$$I_v = \{i \in I \mid \exists r_{vi}\}$$
$$w_{uv} = \frac{|I_u \cap I_v|}{|I_u \cup I_v|} = \frac{|I_u \cap I_v|}{|I_u| + |I_v| - |I_u \cap I_v|}$$

Итак, если $I_u = I_v$, то есть поведение u и v совпадает, $w_{uv} = 1$, а если $I_u \cap I_v = \varnothing$, то есть интересы u и v вообще не пересекаются, $w_{uv} = 0$. То что нужно.

0.40 Задача 2. Как модифицировать AUC-ROC для оценивания качества ранжирования, чтобы в нём сильнее штрафовались нерелевантные документы на верхних позициях?

Дисклеймер. Данное решение является лишь продуктом воображения техера, никем не проверялось на корректность и **может оказаться неверным!**

Вспомним, что AUC-ROC, вообще-то, можно определить не только как площадь под кривой. AUC-ROC – это доля верно упорядоченных пар.

$$\text{AUC-ROC} = \frac{\sum_{i=1}^{q}\sum_{j=1}^{q} I\left[y_i < y_j\right] I'\left[a_i < a_j\right]}{\sum_{i=1}^{q}\sum_{j=1}^{q} I\left[y_i < y_j\right]},$$
$$I'\left[a_i < a_j\right] = \begin{cases} 0, & a_i > a_j, \\ 0.5 & a_i = a_j, \\ 1, & a_i < a_j, \end{cases},$$
$$I\left[y_i < y_j\right] = \begin{cases} 0, & y_i \geq y_j, \\ 1, & y_i < y_j, \end{cases}$$

a_i - ответ алгоритма на i-м объекте, y_i - его истинная релевантность (0 или 1), q - число объектов в тесте.

Перепишем AUC-ROC через количество неправильно упорядоченных пар.

$$\text{AUC-ROC} = 1 - \frac{\sum_{i=1}^{q}\sum_{j=1}^{q} I\left[y_i < y_j\right] I'\left[a_i > a_j\right]}{\sum_{i=1}^{q}\sum_{j=1}^{q} I\left[y_i < y_j\right]}$$

Как можно изменить эту формулу так, чтобы за «нерелевантные документы на верхних позициях» был большой штраф? Логично для этого умножать слагаемые в числителе на какие-нибудь веса, зависящие от позиции

документа в ранжировании, которое выдал алгоритм. То есть в общем виде это будет выглядеть как-то так:

$$M(a,y) = \sum_{i=1}^{q} \left(\sum_{j=1}^{q} I\left[y_i < y_j\right] I'\left[a_i > a_j\right] \right) T\left(\sum_{j=1}^{q} I'\left[a_i > a_j\right] \right)$$

$$\text{AUC-ROC}_{\text{mod}} = 1 - \frac{M(a,y)}{\max\limits_{b} M(b,y)}$$

$T(i) -$ какая-то возрастающая функция

Действительно: для объектов на верхних позициях выдачи алгоритма выражение $\sum_{j=1}^{q} I'\left[a_i > a_j\right]$ будет принимать значения выше, чем для объектов на низких позициях. Это значит, что для объектов на верхних позициях мы за каждую неправильно упорядоченную пару будем штрафовать сильнее, чем для объектов на нижних позициях. А этого мы и добивались.

Дальше можно придумывать конкретные функции T и выводить формулы $\text{AUC-ROC}_{\text{mod}}$ для них, но на коллоквиуме того, что написано выше, должно хватить.

0.41 Задача 3. Как обобщить модель FM, чтобы она учитывала взаимодействия между тройками признаков?

Дисклеймер. Данное решение является лишь продуктом воображения техера, никем не проверялось на корректность и **может оказаться неверным!**

Вспомним, как у нас получилась обычная модель FM (описана в билете 35). Мы хотели строить полиномиальную регрессию над объектами с признаками из $\mathbb{R}^d$:

$$a(x) = w_0 + \sum_{j=1}^{d} w_j x_j + \sum_{j_1=1}^{d} \sum_{j_2=j_1+1}^{d} w_{j_1 j_2} x_{j_1} x_{j_2}.$$

Но решили, что $d(d-1)/2 + d + 1$ параметров – это чёт дофига. Поэтому мы ввели d векторов $v_i \in \mathbb{R}^d$ и решили аппроксимировать $w_{ij} \approx \langle v_i, v_j \rangle$.

$$a(x) = w_0 + \sum_{j=1}^{d} w_j x_j + \sum_{j_1=1}^{d} \sum_{j_2=j_1+1}^{d} \langle v_{j_1}, v_{j_2} \rangle x_{j_1} x_{j_2}.$$

Для такой модели довольно легко придумать модификацию, которая бы учитывала тройки признаков. Введём векторы u_i в дополнение к векторам v_i и построим такую модель:

$$a(x) = w_0 + \sum_{j=1}^{d} w_j x_j + \sum_{j_1=1}^{d} \sum_{j_2=j_1+1}^{d} \langle v_{j_1}, v_{j_2} \rangle x_{j_1} x_{j_2} + \sum_{j_1=1}^{d} \sum_{j_2=j_1+1}^{d} \sum_{j_3=j_2+1}^{d} f\left(u_{j_1}, u_{j_2}, u_{j_3}\right) x_{j_1} x_{j_2} x_{j_3}$$

Здесь f – это какая-то дифференцируемая функция, которая принимает на вход три вектора и выплёвывает число (аналог скалярного произведения для троек).