Конспект по машинному обучению I.

Дмитрий Халанский

6 июня 2018 г.

1.1 Препроцессинг

Препроцессинг. Масштабирование. Нормировка. Полиномиальные признаки. One-hot encoding.

Препроцессинг (*предобработка*). По определению из Википедии: *Data pre-processing is an important step in the data mining process*. Спасибо! Украинская Википедия получше: *Попередня обробка - розділ аналізу даних що займається отриманням характеристик для подальшого використання у наступних розділах аналізу даних.*

Предобработка — это изменение набора данных так, чтобы к нему было легче применять механизмы машинного обучения. Конкретное описание того, какие изменения данных в общем виде можно называть предобработкой, а какие нет, не удалось найти, но правило, кажется, такое: можно осуществлять обратимые преобразования над признаками, добавлять зависящие от старых новые признаки, удалять признаки.

Есть спорный момент: нам говорили, что удалять данные из датасета нельзя никогда. При этом много где (Википедия, какие-то руководства по *науке данных*) говорится про то, что в предобработку входит очистка данных: удаление заведомо некорректных данных, к примеру, "Пол: мужской; Беременность: есть". Более того, говорится про instance selection: выкидывание шумных данных![1] К примеру, допустимо пройти алгоритмом кластеризации и проверить, какие точки никуда не попали.

Масштабирование (normalization) — вид предобработки, который переводит признак в заданный диапазон. К примеру, чтобы перевести в диапазон $[0; 1]$ вектор $\mathbf{x}$, можно применить формулу

$$\mathbf{x}_{\text{new}} = \frac{\mathbf{x} - \min \mathbf{x}}{\max \mathbf{x} - \min \mathbf{x}}$$

[1] `https://en.wikipedia.org/wiki/Instance_selection`

Масштабирование нужно по многим причинам. Некоторые из них таковы:

- Многие алгоритмы за расстояние между двумя точками принимают геометрическое расстояние между ними. Если не осуществлять масштабирование, то некоторые признаки будут вносить в это расстояние заведомо больший вклад.
- Градиентный спуск быстрее сходится, когда данные нормализованы.
- В некоторых методах (к примеру, в softmax) используются операции вида $e^{s_j^L}$, где s_j^L — значение какого-то признака. Без масштабирования такое может быть сложно вычислять с технической точки зрения.

Нормировка (standardization) — преобразование данных к форме, в которой матожидание равно нулю, а среднеквадратическое отклонение (далее — СКО) — единице. Может использоваться для масштабирования.

Формула для нормировки вектора $\mathbf{x}$ такова:

$$\mathbf{x}_{\text{new}} = \frac{\mathbf{x} - \bar{\mathbf{x}}}{\sigma_{\mathbf{x}}}$$

Здесь $\bar{\mathbf{x}}$ — матожидание вектора $\mathbf{x}$, а $\sigma_{\mathbf{x}}$ — его СКО.

Полиномиальные признаки — новые признаки, полученные путём поэлементного перемножения каких-то степеней старых признаков. К примеру, если в исходном наборе данных имелись признаки x, y и z, то полиномиальные признаки степени 3 таковы: x^3, y^3, z^3, x^2y, x^2z, xy^2, y^2z, xz^2, yz^2.

One-hot encoding — способ представления числа в диапазоне $[1; n]$ как набора из n битов, в котором на i позиции стоит единица тогда и только тогда, когда исходное число было равно i. К примеру, если мы рассматриваем диапазон от 1 до 9, то число 6 в такой нотации принимает вид 000100000, а число 8 — вид 010000000.

Этот вид полезен для предобработки признаков, которые задают порядковый номер. Если, к примеру, в признаке содержится 1, 2 или 3 в зависимости от того, зелёный, красный или синий цвет кодируется, то можно разбить данным видом кодирования этот признак на три признака: является ли цвет зелёным, является ли цвет красным и является ли

цвет синим. Это нужно затем, что если мы оставим только один признак и попытаемся предсказать цвет и окажется, что он с равной вероятностью зелёный и синий, то система может сообщить, что цвет красный, а это совсем не то, что нам надо.

1.2 Кластеризация

Кластеризация. kMeans, MeanShift, DBSCAN, Affinity Propagation.

Кластеризация — обнаружение крупных скоплений точек. Более формально, кластеризация — задача назначения точкам из заданного множества классов так, чтобы точки в рамках класса были более "похожи" друг на друга в некотором смысле, чем на точки из других классов.

---

kMeans — алгоритм кластеризации. На вход принимает количество кластеров. Алгоритм выглядит так:

procedure kMeans(clustNo, data)
  $\mathbf{c} \leftarrow$ InitialCenters(clustNo, data) ▷ Начальные центры
  $\mathbf{p} \leftarrow \mathbf{c} + \epsilon$
  while $||\mathbf{c} - \mathbf{p}|| > \epsilon$ **do** ▷ До сходимости:
    $\mathbf{p} \leftarrow \mathbf{c}$
    $x_i \leftarrow \text{argmin}(\mathbf{c} - \text{data}_i)$ ▷ Назначаем классы
    $c_i \leftarrow \text{mean}(\mathbf{data}_{\mathbf{x}=i})$ ▷ Новые центры
  return $\mathbf{c}, \mathbf{x}$.

Неформально: на каждом шаге каждую точку переводим в кластер, центр которого к ней ближе всего, и задаём в качестве нового центра среднее значение координат точек в кластере.

kMeans — простой в реализации алгоритм, который обладает некоторыми недостатками:

- Нужно знать заранее число кластеров;
- Сильно зависит от начального положения центров кластеров;
- Кластеры kMeans по построению всегда сферические;
- Кластеры kMeans имеют один и тот же радиус.

MeanShift — метод кластеризации. Принимает на вход параметр ядра Radial Basis Function, а также максимальный размер кластера. RBF —

функция, которая, в случае гауссова ядра, имеет вид $K(\mathbf{x}) = e^{-c||\mathbf{x}||^2}$ и параметризуется c. Эта функция принимает значение от 1 ($\mathbf{x}$ очень близок к центру) до, в пределе, 0 ($\mathbf{x}$ бесконечно далёк от центра). Таким образом, $K(\mathbf{x} - \mathbf{x}')$ — мера того, насколько близок $\mathbf{x}$ к $\mathbf{x}'$. c определяет, как быстро стремится к нулю эта мера при удалении $\mathbf{x}$ от $\mathbf{x}'$.

На каждом шаге MeanShift производится пересчёт вида $\mu_i \leftarrow m(\mu_i)$, где μ_i — центр i'ого кластера; $m(c)$ определена ниже. $m(\mu_i) - \mu_i$ как раз и называется mean shift.

$$m(\mu_i) = \frac{\sum_{x_j \in N(\mu_i)} K(x_j - \mu_i) x_j}{\sum_{x_j \in N(\mu_i)} K(x_j - \mu_i)}$$

Здесь $N(\mu_i)$ — множество точек в окрестности μ_i. Как раз при вычислении того, принадлежит ли точка окрестности, и используется параметр, задающий максимальный размер кластера.

DBSCAN — ещё один метод кластеризации. Название расшифровывается как Density-Based Spatial Clustering of Applications with Noise.

Параметризуется числами m и ϵ такими, что мы считаем значимыми точки, в ϵ-окрестности которых есть хотя бы m других точек. Такие значимые точки называются объектами плотности (core samples).

Алгоритм сначала ищет все core samples, а затем объединяет в кластер все объекты плотности и их окружения.

Affinity propagation — метод кластеризации. На вход принимает функцию s такую, что $s(i, j)$ — мера близости между i и j. Чем больше $s(i, j)$, тем более близки i и j. Значения функции на диагонали ($s(i, i)$) управляют количеством кластеров: чем выше значение $s(i, i)$, тем выше вероятность, что i будет назначено центром кластера.

Каждой паре точек i и j приписывается *ответственность* r_{ij} и *доступность* a_{ij}. Изначально эти значения 0.

Далее до сходимости происходит **сначала** обновление ответственности:

$$r_{ik} \leftarrow s(i, k) - \max_{j \neq k}(a_{ij} + s(i, j))$$

Затем, после обновления ответственности на всех парах, происходит обновление доступности:

$$f_{ik} = \sum_{j \neq i} \max(0, r_{jk})$$

$$a_{ii} \leftarrow f_{ii}$$
$$a_{ij}, i \neq j \leftarrow \min(0, r_{jj} + f_{ij})$$

Наиболее “ответственные” и “доступные” точки становятся центрами кластеров:

$$\mu_i = \text{argmax}_k(a_{ik} + r_{ik})$$

2.1 Смещение и дисперсия

Смещение и дисперсия (bias and variance). Понятие средней гипотезы.

Пусть мы, имея набор данных D, хотим найти функцию $\hat{f}_D(\mathbf{x})$, которая как можно лучше аппроксимирует функцию $f(\mathbf{x})$. Близость аппроксимации возьмём по методу средних квадратов: требуется минимизировать $\mathbb{E}_{\mathbf{x}}\left[(\hat{f}_D(\mathbf{x}) - f(\mathbf{x}))^2\right]$.

Но D можно рассмотреть как ещё один входной параметр алгоритма. Он появляется, в некотором роде, случайно из функции $f(x)$ с добавлением шума. Таким образом, надо минимизировать ошибку не только по подаваемому значению $\mathbf{x}$, но и по предоставленному в алгоритм набору данных.

$$\mathop{\mathbb{E}}_{D}\left[\mathop{\mathbb{E}}_{\mathbf{x}}\left[(\hat{f}_D(\mathbf{x}) - f(\mathbf{x}))^2\right]\right]$$

Это то же самое, что

$$\mathop{\mathbb{E}}_{\mathbf{x}}\left[\mathop{\mathbb{E}}_{D}\left[(\hat{f}_D(\mathbf{x}) - f(\mathbf{x}))^2\right]\right]$$

Введём понятие *средней гипотезы*: $\hat{f}(x) = \mathbb{E}_D[\hat{f}_D(x)]$ — матожидание модели по набору данных.

Добавим и отнимем внутри скобок $\hat{f}(x)$. Получим

$$\mathop{\mathbb{E}}_{\mathbf{x}}\left[\mathop{\mathbb{E}}_{D}\left[(\hat{f}_D(\mathbf{x}) + \hat{f}(x) - \hat{f}(x) - f(\mathbf{x}))^2\right]\right]$$

Расставим скобки:

$$\mathop{\mathbb{E}}_{\mathbf{x}}\left[\mathop{\mathbb{E}}_{D}\left[((\hat{f}_D(\mathbf{x}) - \hat{f}(x)) + (\hat{f}(x) - f(\mathbf{x})))^2\right]\right]$$

Раскроем квадрат:

$$\begin{aligned}\mathop{\mathbb{E}}_{\mathbf{x}}[\mathop{\mathbb{E}}_{D}[&(\hat{f}_D(\mathbf{x}) - \hat{f}(x))^2 + \\ &(\hat{f}(x) - f(\mathbf{x}))^2 + \\ &(\hat{f}_D(\mathbf{x}) - \hat{f}(x)) \cdot (\hat{f}(x) - f(\mathbf{x}))]]\end{aligned}$$

По линейности матожидания получим

$$\mathop{\mathbb{E}}_{\mathbf{x}}[\mathop{\mathbb{E}}_{D}\left[(\hat{f}_D(\mathbf{x}) - \hat{f}(x))^2\right] + \mathop{\mathbb{E}}_{D}\left[(\hat{f}(x) - f(\mathbf{x}))^2\right] + \mathop{\mathbb{E}}_{D}\left[(\hat{f}_D(\mathbf{x}) - \hat{f}(x)) \cdot (\hat{f}(x) - f(\mathbf{x}))\right]]$$

Заметим, что второе слагаемое не зависит от D. Третье же слагаемое обращается в 0: $\mathbb{E}_D\left[(\hat{f}_D(\mathbf{x}) - \hat{f}(x)) \cdot (\hat{f}(x) - f(\mathbf{x}))\right] = (\hat{f}(x) - f(\mathbf{x})) \cdot \mathbb{E}_D\left[\hat{f}_D(\mathbf{x}) - \hat{f}(x)\right] = (\hat{f}(x) - f(\mathbf{x})) \cdot \left(\mathbb{E}_D[\hat{f}_D(\mathbf{x})] - \hat{f}(x)\right) = (\hat{f}(x) - f(\mathbf{x})) \cdot \left(\hat{f}(x) - \hat{f}(x)\right) = 0$

Получим:

$$\mathop{\mathbb{E}}_{\mathbf{x}}\left[\mathop{\mathbb{E}}_{D}\left[(\hat{f}_D(\mathbf{x}) - \hat{f}(x))^2\right] + (\hat{f}(x) - f(\mathbf{x}))^2\right]$$

Наконец, по линейности матожидания,

$$\underbrace{\mathop{\mathbb{E}}_{\mathbf{x}}\left[\mathop{\mathbb{E}}_{D}\left[(\hat{f}_D(\mathbf{x}) - \hat{f}(x))^2\right]\right]}_{\text{variance}} + \underbrace{\mathop{\mathbb{E}}_{\mathbf{x}}\left[(\hat{f}(x) - f(\mathbf{x}))^2\right]}_{\text{bias}}$$

Таким образом, ошибка на неизвестных данных разбивается на два слагаемых. Первое называется дисперсией, второе — смещением.

Компромисс между смещением и дисперсией — попытка минимизировать суммарную ошибку из двух возможных источников: недообучения и переобучения.

Модели с большой дисперсией и малым смещением хорошо описывают тренировочные данные, но плохо обобщаются на произвольные данные: происходит переобучнние. В то же время модели с малой дисперсией и большим смещением используют недостаточно данных из тренировочной выборки и не отражают имеющиеся паттерны.

Модели с небольшим смещением обычно более сложные и точнее предсказывают тренировочную выборку — в том числе шумовую её составляющую.

2.2 Ансамблевые методы

Ансамблевые методы. Soft and Hard Voting. Bagging. Случайные леса. AdaBoost.

Ансамблевые методы — это методы, опирающиеся на обучение нескольких моделей и агрегирующие ответы от этих моделей. Ансамбли следует использовать, чтобы избежать переобучения: каждая модель будет немного ошибаться на тренировочной выборке, но общие закономерности суммарно в ансамбле можно будет отследить.

Soft and Hard Voting — две стратегии принятия решений ансамблем. Hard voting даёт твёрдый (hard) ответ: однозначно побеждает класс, за который голосует большинство голосов. Soft voting даёт расплывчатый ответ: вычисляется средняя вероятность того, что класс правильный.

Bagging — метод получения нескольких наборов данных, обладая только одним. Заключается в том, что новые наборы данных генерируются путём случайной выборки с повторениями из данного. При этом новые наборы могут быть любого размера, даже больше, чем исходный набор.

Случайные леса — это ансамбли, состоящие из деревьев.

AdaBoost (Adaptive Boosting) — подход к построению ансамблей, который заключается в последовательном создании "слабых гипотез" (которые, по факту, являются просто моделями-классификаторами) так, что каждая последующая слабая гипотеза сосредоточена на исправлении ошибок, допущенных предыдущими слабыми гипотезами.

Алгоритм AdaBoost для бинарного классификатора $X \to \{-1, 1\}$ таков. Сначала присвоим каждому элементу обучающей выборки равный вес ($D_i \leftarrow 1$). Затем до схождения осуществляем следующие шаги:

1. Обучаем слабую гипотезу $h_t : X \to \{-1, 1\}$ так, чтобы взвешенная ошибка была минимальной: $E = \sum_{i=1}^{N} D_i[h_t(x_i) \neq y_i]$, $E \to \min$;
2. Определим вес гипотезы как $\alpha_t \leftarrow \frac{1}{2} \ln\left(\frac{1-E}{E}\right)$;
3. Меняем веса элементов обучающей выборки в зависимости от ошибки новой гипотезы на ней: $D_i \leftarrow D_i e^{-\alpha_t y_i h_t(x_i)}$.

Результатом работы классификатора на $\mathbf{x}$ будет

$$\operatorname{sgn}\left(\sum_{i=1}^{T} \alpha_t h_t(\mathbf{x})\right)$$

3.1 Типы обучения

Типы обучения: с учителем, без учителя, с подкреплением, с частичным участием учителя, активное обучение.

Обучение с учителем (suprvised learning) — обучение, при котором алгоритму обучения на вход подаются данные, а также ожидаемый результат работы обученной системы на каждом экземпляре этих данных. Примерами обучения с учителем являются задачи классификации и регрессии.

Обучение без учителя (unsupervised learning) — обучение, при котором алгоритму на вход подаются только данные и алгоритм должен выявить паттерны в данных. Пример — задача кластеризации.

Обучение с подкреплением (reinforcement learning) — вид обучения, при котором заданы определённые критерии оптимизации ответа системы, но нет привязки конкретных данных к конкретным исходам. Пример — это обучение машины играм: шахматам, шашкам или Доте. Системе сообщают, что поставить мат — очень желанный исход, получить мат — очень нежеланный исход; также желанно убивать фигурки соперника и ставить ему шах; нежелательно давать убивать свои фигурки и получать шах или ничью.

Обучение с частичным участием учителя (semi-suprvised learning) — похоже на обучение без учителя, но некоторым данным присваиваются метки. Пример обучения с частичным участием учителя — составление кластеров, если учитель подсказал некоторые точки, которые точно должны быть в конкретных кластерах.

Активное обучение (active learning) — вид обучения, при котором пользователь интерактивно взаимодействует с системой для совместного обнаружения хорошей модели. Этот вид можно использовать, когда данных без меток много, но при этом вручную каждую точку классифицировать сложно или невозможно. В таком случае система иногда просит у пользователя поставить метки только на небольшом наборе данных. Пример активного обучения — это классификация того, хорошо ли сейчас Google Translate перевёл фразу. Или капча, на которой надо искать дорожные знаки.

3.2 Boosted Decision Trees

Boosted Decision Trees.

Gradient boosting — ансамблевый подход к регрессии и классифика-

ции, который заключается в том, что каждая последующая модель должна исправлять ошибку суммы предыдущих. Иначе, gradient boosting заключается в разложении функции, описываемой тренировочной выборкой, на составляющие, каждая из которых описывается простой моделью.

В общем виде:

$$h_{t+1}(x) \to y - H_t(x)$$

Здесь $H_t(x) = \sum_{i=1}^{t} h_i(x)$, а y — желаемое значение $f(x)$.

Boosted Decision Trees — это gradient boosting на деревьях. В листьях деревьев находятся значения, и при вычислении $H(x)$ осуществляется суммирование значений, лежащих в соответствующих x листьях всех деревьев.

4.1 Ошибки

Ошибка внутри и вне выборки. Ошибка обобщения. Неравенство Хёффдинга. Валидация и кросс-валидация.

Ошибка внутри выборки модели h, также обозначается как $E_{\text{in}}(h)$, — это ошибка, которую допускает модель на элементах из данной выборки.

При данной функции ошибок $e(v, r)$ ошибка внутри выборки модели h имеет вид

$$E_{\text{in}}(h) = \frac{1}{N} \sum_{i=1}^{N} e(h(\mathbf{x_i}), f(\mathbf{x_i}))$$

К примеру, если $e(v, r) = (v-r)^2$, данное выражение становится обычной среднеквадратической ошибкой.

Ошибка вне выборки той же модели — ошибка, которую модель допустит на произвольных элементах. Она имеет вид

$$E_{\text{out}}(h) = \mathop{\mathbb{E}}_{\mathbf{x}} [e(h(\mathbf{x_i}), f(\mathbf{x_i}))]$$

Ошибкой обобщения (generalization error) называется выражение вида $E_{\text{in}}(h) - E_{\text{out}}(h)$. Это мера того, насколько модель h переобучилась, то есть потеряла в общности, подстраиваясь под конкретный набор данных.

Неравенство Хёфдинга (Hoeffding's inequality) утверждает, что

$$P[|\bar{\mathbf{x}} - \mathbb{E}(\bar{\mathbf{x}})| > \epsilon] \leq 2e^{-\epsilon^2 N}$$

где $\mathbf{x}$ — набор независимых случайных величин, N — размер этого набора.

Следствием из этого является такое наблюдение:

$$P[|E_{\text{in}}(h) - E_{\text{out}}(h)| > \epsilon] \leq 2e^{-\epsilon^2 N}$$

Нужно понимать, что это строго выполняется лишь в том случае, когда набор данных, на котором была обучена модель, представляет из себя набор независимых случайных величин.

В целом, это неравенство предоставляет некоторую верхнюю оценку для того, насколько сильно переобученной может быть модель, если ей предоставили набор данных заданного размера.

Валидация — метод оценки E_{out}. Он заключается в том, что из обучающей выборки случайным образом удаляется некоторый набор данных

размера K, который далее называется тестовой выборкой. После обучения модели h вычисляется ошибка E_{val} на тестовой выборке. Формула аналогична формуле для E_{in} и имеет вид

$$E_{\text{val}}(h) = \frac{1}{K} \sum_{i=1}^{K} e(h(\mathbf{x_i}), f(\mathbf{x_i}))$$

Утверждается, что с высокой вероятностью

$$E_{\text{out}}(h) \leq E_{\text{val}}(h) + O\left(\frac{1}{\sqrt{K}}\right)$$

Рекомендованный размер тестовой выборки $K = N/5$.

Кросс-валидация — ещё один метод оценки E_{out}. Он заключается в том, чтобы разбить входные данные на тренировочную и тестовую выборку не одним способом, а многими; на получившемся наборе пар из тренировочной и тестовой выборки провести обучение и валидацию; наконец, взять среднюю ошибку на тестовой выборке как оценку для E_{out}.

Рекомендованный размер тестовой выборки для кросс-валидации $K = N/10$.

4.2 Окололинейные регрессии

Линейная регрессия. Полиномиальная регрессия. Гребневая регрессия.

Линейная регрессия — задача аппроксимации набора данных линейной функцией.

Линейная функция $y = h(\mathbf{x})$ имеет вид $\mathbf{w}^\intercal\mathbf{x}$: это просто скалярное произведение, то есть $y = w_0x_0 + w_1x_1 + \ldots$. Исходя из этого, задача линейной регрессии состоит в нахождении $\mathbf{w}$.

Принимая в качестве ошибки MSE ($e(r, v) = (r-v)^2$), получим ошибку внутри выборки

$$E_{\text{in}}(h) = \frac{1}{N} \sum_{i=1}^{N} (\mathbf{w}^\intercal x_i - y_i)^2 = \frac{1}{N} ||\mathbf{X}\mathbf{w} - \mathbf{y}||^2$$

Воспользуемся знаниями матанализа и заметим, что минимум функции находится в точке со значением производной 0. Перенося это на язык матричного анализа, получим, что минимум этой функции находится в

точке $\nabla E_{\text{in}} = \frac{2}{N}\mathbf{X}^\intercal(\mathbf{X}\mathbf{w} - \mathbf{y}) = 0$. Деля на константу и раскрывая скобки, получим $\mathbf{X}^\intercal\mathbf{X}\mathbf{w} = \mathbf{X}^\intercal\mathbf{y}$. Отсюда имеем

$$\mathbf{w} = (\mathbf{X}^\intercal\mathbf{X})^{-1}\,\mathbf{X}^\intercal\mathbf{y}$$

Так как размерность матрицы $\mathbf{X}$ может быть очень высока, поиск точного ответа по заданной формуле может быть очень ресурсозатратной операцией. В таких случаях имеет смысл искать ответ градиентным спуском.

Полиномиальная регрессия — как линейная регрессия, только с добавлением полиномиальных признаков (см. первый билет).

Регуляризация регрессии — механизм для уменьшения значения ошибки обобщения.[2] Заключается он в том, что дополнительно в ошибку входит с некоторым коэффициентом и сам вектор $\mathbf{w}$, то есть большие значения этого вектора вносят существенный вклад в ошибку. Таким образом, минимизируется выражение

$$E_in(\mathbf{w}) + \frac{\alpha}{N}\mathbf{w}^\intercal\mathbf{w}$$

Градиент этого выражения равен $\frac{2}{N}\mathbf{X}^\intercal(\mathbf{X}\mathbf{w} - \mathbf{y} + \alpha\mathbf{w}) = 0$. Отсюда

$$\mathbf{w} = (\mathbf{X}^\intercal \cdot \mathbf{X} + \alpha\mathbf{I})^{-1}\mathbf{X}^\intercal\mathbf{y}$$

Гребневая регрессия — это поиск $\mathbf{w}$ по формуле выше с заданным коэффициентом α.

---

[2] :)

5.1 VC

Размерность Вапника-Червоненкиса. Размерность Вапника-Червоненкиса для перцептрона.

Размерность Вапника-Червоненкиса $d_{\text{VC}}(H)$ для набора гипотез H — это наибольшее значение n, при котором хоть какой-то набор данных размера n можно разбить на все возможные случаи набором гипотез H.

Это значение равно $k-1$, где k — точка поломки.

Также определение можно перефразировать так: это наибольшее значение n такое, что $m_H(n) = 2^n$, где $m_H(n)$ — функция роста набора гипотез H.

Размерность Вапника-Червоненкиса для перцептрона размерности d равна $d+1$.

Докажем в обе стороны. Сначала докажем, что какой-то набор данных размера $d+1$ можно разбить перцептроном размерности d на все возможные случаи. Заметим, что если у нас есть набор данных $\mathbf{X}$ и набор меток $\mathbf{y}$, то перцептрон корректно классифицирует все данные тогда и только тогда, когда $\text{sgn}(\mathbf{X}^\intercal\mathbf{w}) = \mathbf{y}$. В частности, перцептрон всё правильно разобьёт, если $\mathbf{X}^\intercal\mathbf{w} = \mathbf{y}$. Но найти такое $\mathbf{w}$ можно всегда, когда у $\mathbf{X}$ есть обратная матрица. Тогда просто предоставим матрицу размером $d+1 \times d+1$, в которой первый столбец представляет константу 1, которая имеет обратную. Соответствующий ей набор данных всегда будет разбиваться. Такая матрица есть:

$$\begin{pmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & 1 & 0 & \dots & 0 \\ 1 & 0 & 1 & \dots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 0 & 0 & \dots & 1 \end{pmatrix}$$

Теперь докажем, что никакой набор данных размера $d+2$ разбить перцептроном размерности d. Каждая из этих точек — вектор из $d+1$ числа (включая константу 1 в качестве одного из элементов вектора). Значит, входные данные можно записать как матрицу $d+2 \times d+1$. Её ранг не может превышать $d+1$ (в силу количества столбцов), а значит, среди строк найдётся хотя бы одна линейно зависимая от других. Переставим эту точку так, чтобы она была последней. Тогда $\mathbf{x_{d+2}} = \sum_{i=1}^{d+1} a_i \mathbf{x_i}$. Посмотрим, что будет, когда мы попытаемся эту точку классифицировать. Получится $\text{sgn}(\mathbf{w}^\intercal\mathbf{x_{d+2}}) = \text{sgn}\left(\sum_{i=1}^{d+1} a_i \mathbf{w}^\intercal\mathbf{x_i}\right)$. Теперь добьёмся того, чтобы каждый элемент суммы был положительным. Для этого просто для

всех $i \in [1; d+1]$ назначим $y_i = \text{sgn}(a_i)$. Теперь назначим $y_{d+2} = -1$. Полученное множество нельзя разбить.

5.2

Логистическая регрессия. Градиентный спуск.

Логистическая регрессия — это обнаружение параметров логистической модели. Логистическая модель позволяет не просто оценивать, к какому классу принадлежит данная точка, но и обозначить степень уверенности в данной оценке.

Логистическая функция (сигмоида) имеет вид $\sigma(x) = \dfrac{1}{1+e^{-x}}$. Видно, что у неё есть свойство $\sigma(-x) = 1 - \sigma(x)$.

Параметр логистической модели — как обычно, вектор весов $\mathbf{w}$.

Для обнаружения вероятности того, что точка $\mathbf{x}$ принадлежит классу 1, нужно вычислить $\sigma(\mathbf{w}^\intercal \mathbf{x})$. В этом выражении хорошо видна аналогия σ для логистической регрессии и sgn для перцептрона.

В силу этого и свойства сигмоиды видно, что

$$P(y \mid \mathbf{x}) = \sigma(y\mathbf{w}^\intercal \mathbf{x})$$

Тогда функция правдоподобия принимает вид

$$L(\mathbf{w} \mid \mathbf{x}) = \prod_i P(y_i \mid \mathbf{x_i}) = \prod_i \sigma(y_i \mathbf{w}^\intercal \mathbf{x_i})$$

При логистической регрессии минимизируют значение

$$L(\mathbf{w}) = -N^{-1} \log L(\mathbf{w} \mid \mathbf{x}) = -N^{-1} \sum_i \log \sigma(y_i \mathbf{w}^\intercal \mathbf{x_i}) = N^{-1} \sum_i \log(1+e^{-y_i \mathbf{w}^\intercal \mathbf{x_i}})$$

Обычно $\mathbf{w}$ ищут посредством градиентного спуска. Для этого надо знать производную $\dfrac{\partial L(\mathbf{w})}{\partial \mathbf{w}}$:

$$-N^{-1} \sum_i \frac{y_i \mathbf{x_i}}{1 + e^{y_i \mathbf{w}^\intercal \mathbf{x}}}$$

Градиентный спуск — метод итеративного поиска локальных минимумов. На каждом шаге градиентного спуска происходит обновление с шагом eta оценки значения в ту сторону, в которой наиболее резкий спуск:

$$w \leftarrow w - \eta \frac{\partial C(w)}{\partial w}$$

Стохастический градиентный спуск использует лишь малую часть обучающей выборки при выборе направления спуска.

6.1 Пороговые условия

Пороговые условия. Эффективность по Парето. Precision-Recall и ROC-кривые. AUC.

Пороговые условия — способ преобразовать значение из промежутка в логический результат. К примеру, можно задать пороговое условие, которое преобразует количество восклицательных знаков в письме в указание того, спам ли это.

В общем виде пороговые условия имеют вид

$$R(x) = [a \le x \le b]$$

Синдром пороговых условий — совокупность нескольких пороговых условий; логическое выражение, которое выполняется в том случае, если выполняется достаточное количество пороговых условий.

Имеет вид

$$R(x) = \left[\sum_i [a_i \le x_i \le b_i] \ge d\right]$$

Если d равно количеству пороговых условий, то это просто конъюнкция пороговых условий.

Напоминание, что такое precision и recall:

	$= y$	$\neq y$	
R_+	TP	FP	precision $(R_+ \wedge = y)/(R_+)$
R_-	FN	TN	-
	recall $(R_+ \wedge = y)/(= y)$	fall-out $(R_+ \wedge \neq y)/(\neq y)$	

Правило называется **эффективным по Парето**, если нет правила, у которого выше разом и precision, и recall.

ROC-кривая (Receiver Operating Characteristic) имеет по оси x значение fall-out, по оси y — значение recall.

Precision-Recall-кривая имеет по оси x значение recall, по оси y — значение precision.

AUC (area under curve) — площадь под графиком ROC-кривой.

6.2 Ансамблевые методы регрессии

Ансамблевые методы регрессии. RANSAC. Theil-Sen. Huber.

Ансамблевые методы регрессии — методы аппроксимации функции с использованием набора простых моделей. Более устойчивы к шумам, чем обычный OLS[3].

RANSAC (RANdom SAmple Consensus) — метод ансамблевой регрессии. Состоит в следующем:

1. Гененрируется несколько случайных поднаборов данных;
2. На каждом таком поднаборе обучается отдельная модель;
3. Для каждой модели проверяется, как много точек исходного набора этой моделью предсказывается с некоторой небольшой погрешностью (такие точки называются inliers);
4. Среди тех моделей, которые предсказали достаточное число точек, выбираем лучшую;
5. Дообучаем лучшую модель на оставшемся наборе данных.

Theil-Sen — метод линейной регрессии, который осуществляет линейную регрессию на некоторых подмножествах исходного набора точек, а затем в качестве ответа выбирает медиану полученных векторов $\mathbf{w}$. Устойчив к шумам.

Huber — алгоритм регрессии, который минимизирует квадратную ошибку для inliers (близких точек) и линейную для outliers (дальних точек). Это обусловлено тем, что вблизи к заданной функции требуется

[3]Простите за каламбур.

большая точность: если точка уже далеко, то на неё обращается меньше внимания, она расценивается как шум.

Huber минимизирует функцию

$$\sum_{i}^{N} H_m\left(\frac{x_i w - y_i}{\sigma}\right) + \alpha||\mathbf{w}||^2$$

где

$$H_m(z) = \begin{cases} z^2, & |z| < \epsilon \\ 2\epsilon|z| - \epsilon^2, & |z| \geq \epsilon \end{cases}$$

7.1 Перцептрон

Перцептрон. Перцептрон с карманом.

Перцептрон — линейный классификатор, который обучается, пытаясь последовательно исправлять свои ошибки, встречая новые данные.

Модель заключается в векторе $\mathbf{w}$ весов. Классификация осуществляется так: $h(x) = \text{sgn}(\mathbf{w}^\intercal \mathbf{x})$.

Для обучения перцептрона сначала выбирается произвольный вектор $\mathbf{w}$. Затем в цикле обнаруживается точка $\mathbf{x}$ такая, что перцептрон её неправильно классифицирует, и вектор весов обновляется по правилу $\mathbf{w} \leftarrow \mathbf{w} + y_i \mathbf{x}$.

Перцептрон с карманом — разновидность перцептрона, которая хранит промежуточные результаты и в конце выбирает тот, который дал лучший результат на тестовой выборке. Этот метод позволяет избежать переобучения перцептрона.

7.2 Метод опорных векторов

Метод опорных векторов. Постановка задачи. Формулировка и решение двойственной задачи. Типы опорных векторов. Ядра.

Требуется осуществить линейную классификацию так, чтобы разделяющая линия была как можно дальше от границ классов.

В случае с линейно разделимой выборкой можно найти наиболее близкие друг к другу точки разных классов ($\mathbf{x_i}$ и $\mathbf{x_j}$); тогда максимальная ширина полосы между классами будет равна проекции отрезка между ними на нормаль к разделяющей классы полосе. Вектор нормали назовём $\mathbf{w}$, как и в других задачах линейной классификации.

Если нормализовать $\mathbf{w}$ и смещение b так, что $\min(y_i(\mathbf{w}^\intercal \mathbf{x} - b)) = 1$, то эта проекция будет равна

$$\frac{\mathbf{w}^\intercal(\mathbf{x_i} - \mathbf{x_j})}{||\mathbf{w}||} = \frac{\mathbf{w}^\intercal \mathbf{x_i} - b - (\mathbf{w}^\intercal \mathbf{x_j} - b)}{||\mathbf{w}||} = \frac{2}{||\mathbf{w}||}$$

Нам нужно максимизировать это значение, или минимизировать $||\mathbf{w}||$, или минимизировать $\mathbf{w}^\intercal \mathbf{w}$, если известно, что $y_i(\mathbf{w}^\intercal \mathbf{x} - b) \geq 1$, что то же самое, что $-y_i(\mathbf{w}^\intercal \mathbf{x} - b) + 1 \leq 0$.

Таким образом, имеем задачу оптимизации

$$\begin{cases} \dfrac{1}{2}\mathbf{w}^\intercal\mathbf{w} \to \min \\ -y_i(\mathbf{w}^\intercal\mathbf{x} - b) + 1 \leq 0 \end{cases}$$

Однако если учесть, что выборка может не быть неразделимой, требуется ввести дополнительные ограничения, которые управляют тем, насколько менее предпочтительно допускать нарушение границы, чем сужать разделяющую полосу.

$$\begin{cases} \dfrac{1}{2}\mathbf{w}^\intercal\mathbf{w} + C\sum\limits_{i=1}^{N}\xi_i \to \min \\ -y_i(\mathbf{w}^\intercal\mathbf{x} - b) + 1 - \xi_i \leq 0 \\ \xi_i \geq 0 \end{cases}$$

Здесь ξ_i пропорционально расстоянию, на которое данная точка выходит за границу, отведённую её классу, а C — параметр, который определяет, насколько плохо нарушать границу: чем выше C, тем более строго мы к этому относимся.

По теореме Каруша-Куна-Такера, для систем вида

$$\begin{cases} f(z) \to \min \\ g_i(z) \leq 0 \\ h_i(z) = 0 \end{cases}$$

можно ввести набор переменных α_i^* и β_i^* и определить функцию

$$\mathcal{L}(z, \alpha, \beta) = f(z) + \sum_{i=1}^{m}\alpha_i g_i(z) + \sum_{i=1}^{k}\beta_i h_i(z)$$

так, что

$$\begin{cases} \dfrac{\partial}{\partial z_i}\mathcal{L}(z^*, \alpha^*, \beta^*) = 0 \\ \dfrac{\partial}{\partial \beta_i}\mathcal{L}(z^*, \alpha^*, \beta^*) = 0 \\ \alpha g_i(z^*) = 0 \\ \alpha_i^* \geq 0 \end{cases}$$

Тогда исходная задача сводится к $\max_{\alpha,\beta} \mathcal{L}(z, \alpha, \beta)$.

Видно, что вид нашей задачи полностью совпадает с тем, который требуется по теореме Каруша-Куна-Такера. Применим её. Получим

$$\mathcal{L}(\underbrace{\mathbf{w}, b, \xi}_{z}, \alpha, \beta) = \underbrace{\frac{1}{2}\mathbf{w}^\intercal\mathbf{w} + C\sum_{i=1}^{N}\xi_i}_{f(z)} + \sum_{i=1}^{N}\alpha_i(-y_i(\mathbf{w}^\intercal\mathbf{x_i} - b) + 1 - \xi_i) + \sum_{i=1}^{N}\beta_i\xi_i$$

Перегруппируя скобки,

$$\mathcal{L}(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{1}{2}\mathbf{w}^\intercal\mathbf{w} + \sum_{i=1}^{N}\alpha_i(-y_i(\mathbf{w}^\intercal\mathbf{x_i} - b) + 1) + \sum_{i=1}^{N}\xi_i(C + \beta_i - \alpha_i)$$

Производные по всем переменным из группы z должны быть равны нулю. Тогда

$$\begin{cases} w = \sum \alpha_i y_i \mathbf{x_i} \\ \sum \alpha_i y_i = 0 \\ \alpha_i = C + \beta_i \end{cases}$$

Подставляя эти тождества в исходное выражение, имеем

$$\mathcal{L}(\mathbf{w}, b, \xi, \alpha, \beta) = \underbrace{\frac{1}{2}\sum\sum \alpha_i\alpha_j y_i y_j \mathbf{x_i}^\intercal\mathbf{x_j}}_{\mathbf{w}^\intercal\mathbf{w}} \underbrace{- \sum_{i=1}^{N}\alpha_i y_i \left(\left(\sum \alpha_j y_j \mathbf{x_j}^\intercal\right)\mathbf{x_i} - b\right) + \sum_{i=1}^{N}\alpha_i}_{\sum_{i=1}^{N}\alpha_i(y_i(\mathbf{w}^\intercal\mathbf{x_i} - b) + 1)}$$

$$\mathcal{L}(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{1}{2}\sum\sum \alpha_i\alpha_j y_i y_j \mathbf{x_i}^\intercal\mathbf{x_j} - \sum_{i=1}^{N}\alpha_i y_i \left(\sum \alpha_j y_j \mathbf{x_j}^\intercal\right)\mathbf{x_i} - b\sum_{i=1}^{N}\alpha_i y_i + \sum_{i=1}^{N}\alpha_i$$

$$\mathcal{L}(\mathbf{w}, b, \xi, \alpha, \beta) = \frac{1}{2}\sum\sum \alpha_i\alpha_j y_i y_j \mathbf{x_i}^\intercal\mathbf{x_j} - \sum\sum \alpha_i\alpha_j y_i y_j \mathbf{x_j}^\intercal\mathbf{x_i} + \sum_{i=1}^{N}\alpha_i$$

$$\mathcal{L}(\mathbf{w}, b, \xi, \alpha, \beta) = \sum_{i=1}^{N}\alpha_i - \frac{1}{2}\sum\sum \alpha_i\alpha_j y_i y_j$$

В итоге требуется решить задачу вида:

$$\begin{cases} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum \sum \alpha_i \alpha_j y_i y_j \to \max \\ 0 \le \alpha \le C \\ \sum \alpha_i y_i = 0 \end{cases}$$

В результате имеем набор объектов разных видов:

Внутренние $\alpha_i = 0, \xi_i = 0, y_i(\mathbf{w}^\intercal \mathbf{x} - b) \ge 1$

Опорные $0 < \alpha_i < C, \xi_i = 0, y_i(\mathbf{w}^\intercal \mathbf{x} - b) = 1$

Нарушители $\alpha_i = C, \xi_i > 0, y_i(\mathbf{w}^\intercal \mathbf{x} - b) \le 1$

При расчётах сами значения x_i не используются, важны только их попарные скалярные произведения. Таким образом, чтобы перейти в другое пространство, надо только определить скалярное произведение, не обязательно пересчитывать сами точки.

Ядром называется такая функция $K(x, x')$, что её можно представить в виде $K(x, x') = \psi(x)^\intercal \psi(x')$, где $\psi : X \to H$.

Примеры ядер:

- $K(x, x') = (1 + x^\intercal x)^2$;
- Линейное ядро $\langle x, x' \rangle$;
- Полиномиальное ядро $(r + \gamma \langle x, x' \rangle)^d$;
- Гауссианово ядро (RBF) $e^{-\gamma |x - x'|^2}$;
- Сигмоид: $\mathrm{th}(\sigma \langle x, x' \rangle + r)$.

8.1 Гипотезы и дихотомии

Гипотезы и дихотомии. Функция роста. Точка поломки. Доказательство полиномиальности функции роста в присутствии точки поломки.

Гипотезой называется функция вида $h : X \to C$, то есть классификатор. Для упрощения далее будут рассматриваться только бинарные классификаторы, то есть функции вида $h : X \to \{-1, 1\}$.

Дихотомией на наборе $X' = \{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}\}$ называется элемент множества $\{-1, 1\}^N$, то есть последовательность из N элементов $\{-1, 1\}$. Каждая дихотомия представляет функцию $X' \to \{-1, 1\}$. Комбинаторно очевидно, что дихотомий на наборе из N элементов может быть не более 2^N. Множество дихотомий на наборе X', построенных с использованием гипотез из семейства $\mathcal{H}$, обозначается как $\mathcal{H}(X')$.

Функция роста (growth function) $m_H(N)$ обозначает, какое наибольшее количество различных дихотомий можно построить на N-элементном множестве, используя гипотезы из семейства $\mathcal{H}$.

$$m_H(N) = \max_{\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N}} |\mathcal{H}(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_N})|$$

Точка поломки (breakpoint) для семейства гипотез $\mathcal{H}$ — это наименьшее такое k, что ни для какого набора данных размера k нельзя с использованием гипотез этого семейства получить все возможные дихотомии.

$$\min(k : m_H(k) < 2^k)$$

Определим $B(N, k)$ как максимальное количество дихотомий на наборе из N точек такое, что никакое подмножество размера k не может быть разбито этими дихотомиями. Это комбинаторная величина, которая не зависит от набора гипотез и опирается только на наличие точки поломки k.

Соответственно, если у набора гипотез $\mathcal{H}$ есть точка поломки k, то выполняется $m_H(N) \leq B(N, k)$.

Найдём рекуррентное соотношение для $B(N,k)$. Для этого сначала определим базовые случаи. $B(N,1) = 1$, так как если невозможно разбить даже подмножество из одного элемента, то есть только одна дихотомия: если бы была вторая дихотомия, то она бы отличалась хотя бы в одной точке и подмножество из одного элемента было бы этими двумя дихотомиями разбито. $B(1,k) = 2$ (где $k > 1$), так как подмножества размером k попросту нет, а на одной точке может быть две дихотомии: $\{-1, 1\}$.

Пусть теперь $N \geq 2$ и $k \geq 2$. Рассмотрим все $B(N,k)$ дихотомий. Назовём их множество S. Заметим, что у некоторых дихотомий попарно совпадают все компоненты, кроме последней $(\mathbf{x_1}, \mathbf{x_2}, \ldots, \mathbf{x_{N-1}})$, а последняя различается. Выпишем отдельно все такие пары: те дихотомии, у которых есть пара и компонента $\mathbf{x_N}$ содержит -1, запишем во множество S_2^-, а те, у которых есть пара и компонента $\mathbf{x_N}$ содержит $+1$, запишем во множество S_2^+. Оставшиеся без пары дихотомии запишем во множество S_1. Видно, что $S = S_1 \sqcup S_2^- \sqcup S_2^+$ и $|S_2^+| = |S_2^-|$. Тогда $B(N,k) = |S_1| + 2|S_2^+|$.

Заметим также, что если мы рассмотрим только дихотомии на первых $N-1$ точках, то все их можно получить отбрасыванием последней компоненты дихотомий из S. При этом дихотомии из S_1 войдут один раз, а дихотомии из S_2^+ и S_2^- будут совпадать, так как различаются только в последней компоненте. Значит, $|S_1| + |S_2^+| \leq B(N-1,k)$: раз никакое подмножество размера k от N точек нельзя разбить, то уж тем более никакое подмножество размера k от $N-1$ точки тоже нельзя разбить.

Также никакое подмножество первых $N-1$ точек размера $k-1$ нельзя разбить дихотомиями из множества S_2^+, поскольку в противном случае можно было бы просто добавить в набор точку $\mathbf{x_N}$, которую можно разбить дихотомиями из S_2^+ и S_2^-. Значит, $|S_2^+| \leq B(N-1,k-1)$.

Получается, $B(N,k) = |S_1| + |S_2^+| + |S_2^+| \leq B(N-1,k-1) + B(N-1,k)$.

Осталось доказать, что $B(N,k) \leq \sum_{i=0}^{k-1} \binom{N}{i}$. Доказывается это по индукции. Для базовых случаев это тривиально верно. В качестве шага индукции заметим

$$
\begin{aligned}
B(N,k) &\le B(N-1,k-1) + B(N-1,k) \\
&\le \sum_{i=0}^{k-2} \binom{N-1}{i} + \sum_{i=0}^{k-1} \binom{N-1}{i} \\
&= \sum_{i=1}^{k-1} \binom{N-1}{i-1} + \sum_{i=0}^{k-1} \binom{N-1}{i} \\
&= \sum_{i=1}^{k-1} \binom{N-1}{i-1} + \sum_{i=1}^{k-1} \binom{N-1}{i} + 1 \\
&= \sum_{i=1}^{k-1} \left(\binom{N-1}{i-1} + \binom{N-1}{i} \right) + 1 \\
&= \sum_{i=1}^{k-1} \binom{N}{i} + \binom{N}{0} \\
&= \sum_{i=0}^{k-1} \binom{N}{i}
\end{aligned}
$$

8.2 Деревья решений

Деревья решений. Информационный выигрыш, критерий Джини. Регуляризация деревьев. Небрежные решающие деревья.

Деревья решений — модели на основе деревьев. В вершинах дерева записаны правила, по которым можно понять, в какую вершину спускаться дальше с такими входными данными. В листьях записаны ответы.

В целом, дерево — способ представления функции, значения которой лежат в листьях. Промежуточные вершины дерева делят область входных значений.

Критерий информационного выигрыша $\mathrm{IG}(R)$ сообщает изменение в суммарной энтропии системы, если мы разобьём набор входных данных по правилу R:

$$
\mathrm{IG}(R) = H(X) - \frac{|R^1|}{|X|} H(R^1) - \frac{|R^2|}{|X|} H(R^2)
$$

Здесь $H(S)$ — энтропия множества S.

Критерий Джини (Gini impurity) имеет вид

$$I_g(X) = \sum_{y \in Y} \frac{|x_i : y_i = y|}{|X|} \frac{|x_i : y_i \neq y|}{|X|}$$

Используется вместо энтропии и вместо количества информации сообщает вероятность ошибки при приписывании элементам множества класса согласно распределению классов во множестве. Это мера неоднородности множества.

Регуляризация деревьев — процедура подбора параметров деревьев, таких как глубина. Для регуляризации выделяется выборка test, не участвующая в обучении дерева, по результатам на которой оценивается качество дерева.

Небрежные решающие деревья — деревья, в которых на одном уровне одинаковые признаки.

В Yandex по ошибке сделали новый вид небрежных решающих деревьев MatrixNet. В этих деревьях правило разделения на каждом уровне тоже одинаковое.

9.1 Байесовский классификатор

Байесовский классификатор. Типы оценки распределений признаков (Gaussian, Bernoulli, Multinomial). EM-алгоритмы.

Байесовский классификатор — классификатор, который работает, исходя из предположения, что признаки взаимно независимы. Это предположение почти никогда не является верным, поэтому байесовский классификатор можно использовать как удобную точку отсчёта при оценке классификаторов: если не удалось победить байесовский классификатор, значит, алгоритм не смог никакой информации получить из взаимозависимых признаков.

Предположение о независимости более формально можно записать так:

$$P(x_1, x_2, \ldots, x_n \mid y) = P(x_1 \mid y) \cdot P(x_2 \mid y) \cdots P(x_n \mid y)$$

Байесовский классификатор основан на теореме Байеса, которая утверждает, что

$$P(y \mid x) = \frac{P(x \mid y) \cdot P(y)}{P(x)}$$

Словами: вероятность того, что точка x принадлежит классу y, равна вероятности того, что класс y может содержать точку x, умножить на долю класса y, делить на вероятность появления точки x.

Для классификации объекта надо найти класс, которому данный объект принадлежит с наибольшей вероятностью. Из формул выше,

$$y_{\text{MAP}} = \text{argmax}_{y \in Y} \frac{P(x \mid y) \cdot P(y)}{P(x)} = \text{argmax}_{y \in Y} P(x \mid y) \cdot P(y)$$

Гауссова оценка $P(x_i \mid y) = \dfrac{1}{\sqrt{2\pi(\sigma_i^y)^2}} e^{-\dfrac{(x_i - \mu_i^y)^2}{2(\sigma_i^y)^2}}$;

Оценка Бернулли $P(x_i \mid y) = P(x_i = 1 \mid y)x_i + (1 - P(x_i = 1 \mid y))(1 - x_i)$ (предполагается, что $x_i \in \{-1, 1\}$);

Мультиномиальная $P(x_i \mid y) = \dfrac{\text{count}(x_i, y) + \alpha}{\text{count}(y) + \alpha K}$.

Expectation-maximization — итеративный алгоритм для обнаружения maximum a postriori в сложных распределениях.

Алгоритм состоит из двух фаз: E (Expectation) и M (Maximization). На фазе E происходит вычисление принадлежности объектов разным классам (заданным распределениями), а на фазе M происходит перерасчёт соответствующих классам распределений.

Рассмотрим случай, когда распределения заданы k гауссианами.

У системы имеются параметры μ_k (вектор среднего), Σ_k (матрица ковариаций) и α_k (мера того, сколько данному распределению принадлежит точек). $\sum_i \alpha_i = 1$.

Принадлежность объекта x_i к k'ому распределению (фаза E) вычисляется по формуле

$$w_{ik} = p(\mu_k, \Sigma_k \mid x_i) = \frac{p(x_i \mid \mu_k, \Sigma_k) a_k}{\sum_j p(x_i \mid \mu_j, \Sigma_j) a_j}$$

Перерасчёт параметров с учётом принадлежностей (фаза M) осуществляется по формулам:

$$\begin{aligned}
N_k &\leftarrow \sum_{i=1}^{N} w_{ik} \\
\alpha_k &\leftarrow \frac{N_k}{N} \\
\mu_k &\leftarrow \frac{1}{N_k} \sum_{i=1}^{N} w_{ik} x_i \\
\Sigma_k &\leftarrow \frac{1}{N_k} \sum_{i=1}^{N} w_{ik} (x_i - \mu_k)(x_i - \mu_k)^\intercal
\end{aligned}$$

9.2 Нейронные сети

Нейронные сети. Перцептрон Розенблатта. Функции активации. Обратное распространение градиента. Softmax.

Нейронные сети — класс алгоритмов машинного обучения, в которых фигурируют преобразующие входные данные нейроны и взвешенные связи между ними. В нейронной сети на вход каждому нейрону поступает на вход линейная комбинация выходов из предыдущего слоя; к этому

входу применяется какая-то **функция активации**, и результат становится выходным значением данного нейрона.

Перцептрон Розенблатта — перцептрон со структурой нейронной сети. Как и в обычном перцептроне, признаки на входе суммируются с некоторым весом и проверяется знак результата, но здесь сумматор и операция взятия знака представлены в виде нейронов, а вектор весов перцептрона становится просто набором весов между входными нейронами и сумматором.

Функция активации нейрона — функция, которую нейрон применяет ко входу для получения выхода.

Рассматривались такие функции активации:

- Сигмоид $\sigma(s) = (1 + e^{-s})^{-1})$;
- Tanh $\sigma(s) = (e^s - e^{-s})/(e^s + e^{-s})$;
- ReLU (нелинейное преобразование) $\sigma(s) = \max(0, s)$.

Backpropagation — обновление предыдущих слоёв нейронной сети на основании изменений в последующих.

Для этого на последнем слое происходит, как обычно, градиентный спуск. При этом произошло обновление вектора весов на последнем слое по правилу $\mathbf{w^L} \leftarrow \mathbf{w^L} - \eta x_i^{L-1} \delta^L$, просто по виду градиентного спуска.

На каждом шаге теперь будем высчитывать δ^{l-1}, зная δ^l, по формуле $\sum \delta_i^l \times w_{ij}^l \times \sigma'(s_i^{l-1})$, и обновлять вектор весов $\mathbf{w^l}$.

Softmax — функция для классификации с использованием нейронных сетей. Имеет вид

$$x_j^L = \sigma^L(s_j^L) = \frac{e^{s_j^L}}{\sum e^{s_i^L}}$$

На выходе каждого нейрона содержится вероятность принадлежности входных данных соответствующему классу. Сумма вероятностей по всем выходам, что видно и по форме функции, равна 1.

10.1 Стохастическая оптимизация

Стохастическая оптимизация. Hill Climb. Отжиг. Генетический алгоритм.

Стохастическая оптимизация — вероятностное обнаружение минимумов/максимумов функции. Если имеется возможность продифференцировать функцию, применим градиентный спуск. Иначе нужно осуществлять поиск, локальный или глобальный.

Hill climbing — алгоритм локального поиска, который работает как градиентный спуск: на основании значений функции в ближайших точках выбирается направление подъёма.

Stochastic hill climbing выбирает направление с некоторой зависящей от значения функции вероятностью.

Tabu search не возвращается в недавно пройденные точки.

Particle swarm optimization осуществляет обход из нескольких точек сразу, и направление последующего подъёма каждой точки определяется как линейная комбинация лучшего подъёма среди локальных для неё и направления к наибольшему значению из уже известных хоть какой-то точке.

Отжиг (annealing) — алгоритм локального поиска. Для него вводится понятие температуры. Температура — это мера того, с какой вероятностью алгоритм пойдёт в направлении, которое не улучшает значение оптимизируемой функции. С каждой итерацией температура убывает и, следовательно, реже происходит переход, не приводящий к улучшению функции.

Отжиг используется для борьбы с локальными максимумами: пока температура не сильно упала, есть шанс уйти из локального максимума в другую позицию, откуда доступен более хороший максимум.

Генетический алгоритм — алгоритм локального поиска, который хранит в каждый момент времени популяцию из возможных решений задачи оптимизации и на каждом шаге выбирает два лучших решения, каким-то образом их скрещивает и возвращает потомков в популяцию, заменяя ими два худших решения. Также генетический алгоритм может

выполнять случайные мутации или на потомках, или на всей популяции, или совсем случайно.

10.2 Метрические классификаторы

Метрические классификаторы. kNN. WkNN. Обтор эталонов. DROP5. Kdtree.

Метрические классификаторы — это модели, которые позволяют узнавать класс, в которому наиболее близка данная точка, если уже известны классы других точек.

В общем виде метрический классификатор имеет форму

$$h(x; D) = \text{argmax}_{y \in Y} \underbrace{\sum_{x_i \in D} [y_i = y] \underbrace{w(x_i, x)}_{\text{близость к соседу } x_i}}_{\text{близость к классу } y}$$

---

kNN (k nearest neighbors) — метрический классификатор, который назначает $w(x_i, x) = [x_i$ — один из k ближайших соседей$]$.

WkNN (weighted kNN) — похожий на kNN классификатор, но k ближайшим соседям назначен вес не 1, а как-то зависящий от расстояния до них. К примеру, могут быть веса $w(x_i, x) = \dfrac{r - \rho(x, x_i)}{r}$, где r — параметр, а rho — расстояние; или $w(x_i, x) = q^{-\rho(x, x_i)}$, где q — параметр.

Отступ (margin) точки x_i — это мера того, насколько данная точка окружена точками своего же класса. Определяется как разность близости данной точки к своему классу и наибольшей близости к чужому классу.

Точки с большим отступом надёжно классифицируются; точки с наибольшими значениями отступа можно выбрать как **эталонные объекты**.

Обзор эталонов (prototype selection) — выбор эталонных объектов с целью упрощения процедуры классификации. Если выбрано множество эталонных объектов Ω, то классификация точки x осуществляется не как $h(x; D)$, а как $h(x; \Omega)$.

Существует много методов отбора эталонов. Они делятся по направлению поиска (инкрементальные, декрементальные, пакетные, смешанные и так далее) и по типу выбора (сжатие, изменение, гибридный).

DROP5 — очень плохо гуглящийся алгоритм обзора эталонов.

Заключается в том, что точки сортируются по возрастанию расстояния до ближайшей точки другого класса, а затем по ним осуществляется обход, и точка P удаляется, если все точки, для которых она была ближайшей в исходном наборе, и без неё правильно классифицируются.

---

k-d tree — алгоритм, позволяющий в многомерных пространствах быстро находить соседей. Для этого пространство разбивается плоскостями, каждая из которых перпендикулярна какой-то оси координат, причём каждый раз разбиение осуществляется так, чтобы в каждой половине оставалось поровну точек (граница проходит через медиану).

Если теперь нужно k соседей, то сначала рассматриваются все соседи в текущем листе, и если их не хватает, алгоритм поднимается на уровень выше, и так пока не наберётся k соседей.