

ML 2017

3 курс ФИВТ, 2017

9 июня 2017 г.

Содержание

I Билеты	3
1 Билет 1	3
1.1 Supervised и unsupervised learning	3
1.1.1 Supervised learning	3
1.1.2 Unsupervised learning	4
1.2 Стандартные задачи	4
1.2.1 Задача классификации	4
1.2.2 Задача регрессии	4
1.2.3 Задача кластеризации	4
1.3 Простые модели	5
1.3.1 kNN	5
1.3.2 Naive Bayes	5
1.3.3 Linear Regression	5
1.3.4 kMeans	5
1.4 Оценка качества	6
1.4.1 Кросс-валидация	6
1.4.2 Кривые обучения	6
1.5 Переобучение и недообучение, как их детектировать	6
1.5.1 Переобучение	6
1.5.2 Недообучение	6
1.5.3 Как детектировать	6
1.6 Извлечение признаков	6
1.6.1 Текст	6
1.6.2 Изображение	6
1.6.3 Звук	7
1.7 Предобработка признаков	7
1.7.1 Разреженные признаки	7
1.7.2 Категориальные признаки	7
2 Метрики качества в задачах классификации и регрессии: accuracy, precision, recall, F1, ROC-AUC, log loss, MSE, MAE, квантильная, MAPE, SMAPE. Когда какая из метрик предпочтительней. К оценке каких параметров распределений ответов приводят log loss, MSE и MAE (с обоснованием).	7
2.1 Метрики качества в задачах классификации.	7
2.1.1 Accuracy	7
2.1.2 Precision	7
2.1.3 Recall	8
2.1.4 F-мера	8
2.1.5 ROC-AUC	8
2.2 Метрики качества в задачах регрессии.	9
2.2.1 LogLoss	9
2.2.2 MSE	9
2.2.3 MAE	9
2.2.4 MAPE	9
2.2.5 SMAPE	10

3 Решающие деревья. Как работает уже построенное решающее дерево. Рекурсивное построение деревьев. Энтропийный критерий и критерий gini. Классификационные и регрессионные деревья: в чем различия. Настройка гиперпараметров решающего дерева. Преимущества и недостатки деревьев.	10
3.1 Работа решающего дерева	10
3.2 Как строятся решающие деревья?	10
3.3 Как выдавать ответ в листе?	10
3.4 Как выбрать наилучшее разбиение?	11
3.5 Настройка гиперпараметров	11
3.6 Преимущества и недостатки деревьев.	11
4 Общие методы построения композиций - блэндинг, стэкинг, бэггинг и бустинг. Bias-variance trade-off (без вывода). Анализ бустинга и бэггинга с помощью bias-variance trade-off.	12
4.1 Bagging = Bootstrap aggregation	12
4.2 Stacking	12
4.3 Blending	12
4.4 Bias-variance trade-off	12
5 Бэггинг и Random Forest. Связь корреляции между ответами моделей и качеством модели в бэггинге.	13
5.0.1 Бэггинг.	13
5.0.2 Random Forest.	13
6 Бустинг и GBM. Выбор параметров в ансамблях решающих деревьев. Сравнение Random Forest и GBDT (подбор параметров, переобучение).	14
6.1 Gradient boosting	14
6.2 Обучение базовых алгоритмов в GB	15
6.3 Описание алгоритма градиентного бустинга	16
6.4 Градиентный бустинг для регрессии	16
6.5 Градиентный бустинг для классификации	16
6.6 Градиентный бустинг для решающих деревьев	17
7 Линейные модели в задачах классификации и регрессии: функции потерь, регуляризаторы, оптимизационные задачи. Стохастический градиентный спуск.	17
8 Линейная регрессия. Геометрический и аналитический вывод формулы для весов признаков. Регуляризация в линейной регрессии: гребневая регрессия и LASSO.	19
9 Логистическая регрессия. Варианты записи оптимизационной задачи. Оценка вероятности принадлежности к классу. Настройка параметров с помощью стохастического градиентного спуска.	21
10 Метод опорных векторов: оптимизационная задача в условной и безусловной форме. Опорные векторы (достаточно записать и продифференцировать Лагранжиан, оптимизационная задача, выраженная через двойственные переменные - опционально). Идея Kernel Trick.	22
11 Нейронные сети, обучение (backprop), слои для нейронных сетей (dense, conv, pooling, batchnorm), нелинейности (relu vs sigmoid, softmax), функции потерь (logloss, l2, hinge)	24
11.1 Обучение нейронных сетей	24
11.2 Слои для нейронных сетей	25
11.3 Нелинейности (relu vs sigmoid, softmax)	28
11.4 Функции потерь (logloss, l2, hinge)	29
12 Нейронные сети, обучение (backprop), оптимизация для нейронных сетей (sg, msg, nmsg, rmsprop, adam), регуляризация нейросетей (dropout, dropconnect, l1, l2, batchnorm)	29
12.1 Оптимизация:	29
12.2 Регуляризация	30
13 Нейронные сети, обучение (backprop), современные сверточные нейронные сети (vgg, resnet, inception) и детали обучения (batchnorm, pretraining)	31
13.1 VGG	31
13.2 Inception	32
13.3 ResNet	32

14 Рекуррентные НС, обучение (backprop тт), отличие от сверточных, разновидности рекуррентных слоев (RNN, LSTM, GRU) аннотация изображений, перевод, диалоговые системы	33
14.1 Рекуррентные НС	33
14.2 Backprop ТТ	33
14.3 Vanishing Gradient	33
14.4 LSTM	34
14.5 Аннотация изображений, перевод, диалоговые системы	36
15 Задача снижения размерности пространства признаков. Идея метода главных компонент (PCA). Связь PCA и сингулярного разложения матрицы признаков (SVD).	36
16 Вычисление SVD в пространствах высокой размерности методом стохастического градиента (SG SVD).	37
17 Идея методов SNE, tSNE, принципиальные отличия от PCA.	37
18 Задача кластеризации. Агglomerативная и дивизионная кластеризация.	38
19 Кластеризация с помощью EM-алгоритма (без вывода М-шага). Формула Ланса-Уилльямса.	38
20 Алгоритмы k-Means и DBSCAN.	39
 II Доп. вопросы НЕ ПИШИТЕ ИХ	40
21 Основные понятия	40
22 Простые методы	40
23 Метрики качества в задачах классификации и регрессии	40
24 Деревья	40
25 Общие идеи построения композиций	41
26 Градиентный бустинг	41
27 Случайный лес	41
28 Linear models	41
29 Neural networks	42
30 Unsupervised learning	43
31 Теоретический минимум	43

Часть I

Билеты

1 Билет 1

1.1 Supervised и unsupervised learning

1.1.1 Supervised learning

Формально: Пусть есть множество *объектов* X , множество *допустимых ответов* Y и существует *целевая функция* $y^* : X \rightarrow Y$ значения которой известны только на конечном множестве объектов $\{x_1, \dots, x_l\} = X^l \subset X$. При этом объекты описываются *признаками*, то есть есть n функций $f_1, \dots, f_n, f_i : X \rightarrow D_i$, значения которых нам известны на тестовой выборке. Соответственно нужно построить *решающую функцию* $a : D_1 \times D_n \rightarrow Y$, которая «хорошо» бы приближала y^* на всем X .

Суть: для обучающей выборки есть ответы.

Вода: Обучение с учителем (англ. Supervised learning) — один из способов машинного обучения, в ходе которого испытуемая система принудительно обучается с помощью примеров «стимул-реакция». Известна только конечная совокупность прецедентов — пар «стимул-реакция», называемая обучающей выборкой. На основе этих данных требуется восстановить зависимость (построить модель отношений стимул-реакция, пригодных для прогнозирования), то есть построить алгоритм, способный для любого объекта выдать достаточно точный ответ. Для измерения точности ответов, так же как и в обучении на примерах, может вводиться функционал качества.

1.1.2 Unsupervised learning

Суть: для обучающей выборки нет ответов.

Вода: Обучение без учителя (самообучение, спонтанное обучение) — один из способов машинного обучения, при котором испытуемая система спонтанно обучается выполнять поставленную задачу без вмешательства со стороны экспериментатора. Как правило, это пригодно только для задач, в которых известны описания множества объектов (обучающей выборки), и требуется обнаружить внутренние взаимосвязи, зависимости, закономерности, существующие между объектами.

1.2 Стандартные задачи

1.2.1 Задача классификации

Формально: Y — конечное множество. Часто $Y = \{1, \dots, M\}$ или Y — неупорядоченно. Если $|Y| = 2$, то говорят о *бинарной классификации*.

Примеры:

- Задача медицинской диагностики. Объекты — пациенты; признаки — пол, возраст, иные данные о физическом состоянии / обстановке, результаты обследований, выданные препараты, etc; целевая функция — диагнозы.
- Задача кредитного скоринга — Объекты — заемщики; признаки — пол, возраст, иные данные о физическом состоянии / обстановке, доход, имущество, семейное положение, кредитная история, etc; целевая функция — давать кредит или нет (бинарная классификация).
- и еще миллиард задач

Вода: Задача классификации — формализованная задача, в которой имеется множество объектов (ситуаций), разделённых некоторым образом на классы. Задано конечное множество объектов, для которых известно, к каким классам они относятся. Это множество называется выборкой. Классовая принадлежность остальных объектов неизвестна. Требуется построить алгоритм, способный классифицировать (см. ниже) произвольный объект из исходного множества.

1.2.2 Задача регрессии

Формально: $Y = \mathbb{R}$ (или Y — промежуток из \mathbb{R} . Или не промежуток. Вообще с формализмом тут тугу.

Примеры:

- Задача прогнозирования спроса. Объекты — товары; признаки — да что угодно: вес, цвет, запах, вкус, удобство использования...; целевая функция — сколько людей купят товар через M единиц времени.
- Задача предсказания роста сына по росту отца. Объекты — люди; признаки — отклонение роста от среднего; целевая функция — отклонение роста сына от среднего. [Гальтон исследовал, формула $y = \frac{2}{3}x$ неплохо работает]

Вода: Есть множество объектов. Есть функция на нем. Для некоторого подмножества объектов задано значение функции. Нужно научиться предсказывать значение функции на других объектах. Качество выполнения задачи определяется функционалом качества. Например MSE.

1.2.3 Задача кластеризации

Задача кластеризации (unsupervised or semi-supervised) заключается в следующем. Имеется обучающая выборка $X^l = \{x_1, \dots, x_l\} \subset X$ и функция расстояния между объектами $\rho(x, x')$. Требуется разбить выборку на непересекающиеся подмножества, называемые кластерами, так, чтобы каждый кластер состоял из объектов, близких по метрике ρ , а объекты разных кластеров существенно отличались. При этом каждому объекту $x_i \in X_l$ приписывается метка (номер) кластера y_i

1.3 Простые модели

1.3.1 kNN

Кратко:

Задача: Пусть на множестве объектов X задана (ну или мы ее выдумали) функция расстояния $\rho : X \times X \rightarrow [0, \infty)$ [обычно удовлетворяющая свойствам метрики: 1) $\rho(x, y) = 0 \Leftrightarrow x = y$ 2) $\rho(x, y) = \rho(y, x)$ 3) $\rho(x, z) \leq \rho(x, y) + \rho(y, z)$]. Требуется решить задачу классификации или регрессии.

Классификация.

Выберем для всех функцию $\omega : X \times X \rightarrow \mathbb{R}$ — функция оценки важности одного объекта для классификации другого.

$$\text{Тогда } a(x) = \arg \max_{y \in Y} \sum_{i=1}^l I(y_i = y) \omega(x_i, x)$$

В простейшем методе k ближайших соседей берем $\omega(x, y) = I\left(\left\{\sum_{i=1}^l I(\rho(x_i, x) < \rho(y, x))\right\} \leq k\right)$ (то есть если y — один из k ближайших соседей x , то $\omega = 1$, иначе 0).

Можно также брать $\omega = F(\rho(x, y))$ если y — один из k ближайших соседей x , иначе 0), где $F(\rho)$, например, задана как $\frac{1}{1+\rho}$ или $F(\rho) = -\rho$.

Подробно:

[kNN](#) (с 43)

1.3.2 Naive Bayes

Кратко:

Будем рассматривать $X \times Y$ как вероятностное пространство с плотностью распределения $p(x, y) = P(y) \cdot p(x|y)$. $P_y := P(y)$ называется *априорной вероятностью классов*, плотности $p_y(x) := p(x|y)$ называются *функциями правдоподобия классов*.

Баесовский классификатор: $a(x) = \arg \max_{y \in Y} p(y|x) = \arg \max_{y \in Y} \frac{P(y)p(x|y)}{P(x)} = \arg \max_{y \in Y} P(y)p(x|y)$.

На самом деле вместо $P(y)$ и $p(x|y)$ используются оценки. Оценить $P(y)$ легко частотой класса в обучающей выборке. Оценить $p(x|y)$ сложнее (потому что оно многомерное).

Если принять гипотезу независимости признаков, то $p(x|y) = p_1(x^{(1)}|y) \cdot p_2(x^{(2)}|y) \cdots p_n(x^{(n)}|y)$, где $p_i(x^{(i)}|y)$ — условная вероятность i -го признака. [на самом деле вроде нет, надо спросить на консе]

$p_i(x^{(i)})$ уже можно неплохо оценить (какими-нибудь статистическими методами, типа предположить что оно из какого-то семейства (нормальное, бернуlli, мультиномиальное, ...)) и взять оценку максимального правдоподобия)

Это и есть наивный баесовский классификатор.

Подробно:

[Naive Bayes](#) (с 21)

1.3.3 Linear Regression

Кратко:

Перед нами стоит задача регрессии. Будем строить $a(x)$ в виде $a(x) = \langle w, x \rangle + w_0$ или, добавив $x_i^{(0)} = 1$, $a(x) = \langle w, x \rangle$.

Пусть F — матрица признаков ($F_{ij} = x_i^{(j)}$), \vec{y} — вектор ответов, \hat{y} — вектор предсказаний ($\hat{y}_i = a(x_i) \Rightarrow \hat{y} = Fy$).

Если искать w , минимизируя квадратичную ошибку ($w = \arg \min ||\hat{y} - y||^2$, получим $w = (F^T F)^{-1} F^T y$).

Над этим можно извращаться миллиардом разных способом, подробности — в соответствующих билетах.

Выход формулы:

Точка минимума — стационарная \Rightarrow все частные производные равны нулю. Считаем ручками, сравниваем с формулой сверху, радуемся.

Подробно:

[Linear Regression](#) (с 84)

1.3.4 kMeans

Кратко:

Решаем задачу кластеризации на k классов в метрическом пространстве.

Алгоритм:

1. Выбираем инициализацию центров кластеров (случайно или k самых удаленных кластеров). μ_y — центр кластера y
2. Повторяем, пока алгоритм не сойдется (пока y_i меняются):

(a) Отнести каждый объект к ближайшему центру $y_i := \arg \min_{y \in Y} \rho(\mu_y, x_i)$

(b) Вычислить новое положение центров как среднее в своем классе: $\mu_y = \frac{\sum I(y_i=y)x_i}{\sum I(y_i=x)}$

Подробно:

k-Means (с 120)

1.4 Оценка качества

1.4.1 Кросс-валидация

Перекрёстная проверка (англ. cross-validation) — метод оценки аналитической модели и её поведения на независимых данных. При оценке модели имеющиеся в наличии данные разбиваются на k частей. Затем на $k - 1$ частях данных производится обучение модели, а оставшаяся часть данных используется для тестирования. Процедура повторяется k раз; в итоге каждая из k частей данных используется для тестирования. В результате получается оценка эффективности выбранной модели с наиболее равномерным использованием имеющихся данных.

1.4.2 Кривые обучения

Кривые на графике ошибки, показывающие качество обучения в зависимости от размера тестовой выборки или каких-либо параметров модели.

1.5 Переобучение и недообучение, как их детектировать

1.5.1 Переобучение

Переобучение - явление, когда построенная модель хорошо работает на объектах из тестовой выборки, но плохо работает на объектах, не участвовавших в тестовой выборке.

1.5.2 Недообучение

Недообучение связано с тем, что по каким-то причинам алгоритм не уловил закономерностей в данных. Это явление, обратное переобучению, при котором алгоритм не полностью использует предоставленные ему для обучения данные.

1.5.3 Как детектировать

Посмотреть кривые обучения для тестовой и тренировой выборки (кросс-валидация). Если кривые почти совпадают и растут, значит недообучение. Если кривая тестовой выборки уходит вниз от кривой тренировой выборки, значит переобучение.

Это лишь мои мысли, они могут быть неправильные, проверьте.

1.6 Извлечение признаков

Подробнее можно почитать [тут](#)

1.6.1 Текст

Можно выделить n -граммы слов, встречающиеся в текстах и сделать i -й признак — количество i -х n -грамм в тексте.

Можно приводить слова в начальную форму (стемминг — эмпирический отброс суффикса или лемматизация — постановка слова в начальную форму).

Можно использовать n -граммы букв, чтобы учесть различные словоформы.

Можно использовать стоп-лист — игнорировать слишком частые слова типа *a, the, and, is, ...*

Можно использовать не количество слов, а так называемый $TF-IDF : Term frequency - inverse document term frequency$.

Смысл в том, чтобы более частым словам приписывать меньший вес.

$TF(t, d) = \frac{n_t}{\sum n_k}$, где n_t — число вхождений слова k в документ d .

$IDF(t) = \log \frac{|D|}{\#\{d: t \in D\}}$, где t — слово, D — множество документов.

$TF-IDF = TF(t, d) \cdot IDF(t)$

1.6.2 Изображение

- Координаты углов, границ областей
 - Цвет (среднее значение пикселя)
 - Переходы
 - использование сетей для feature extraction

1.6.3 Звук

- MFCC - преобразование Фурье логарифма спектра

1.7 Предобработка признаков

1.7.1 Разреженные признаки

Берем меньшее количество признаков, сопоставляем старым признакам новые (например, $i \mapsto i \% d$, где d — количество новых признаков. И новый признак — сумма соответствующих старых.

Можно уменьшить размерность (см бытет про PCA).

1.7.2 Категориальные признаки

Простейший способ - каждой категории сопоставить некоторое число (например, номер признака или хэш).

Ещё один способ: для кодируемого категориального признака создаётся N новых признаков, где N — число категорий. Каждый i -й признак — бинарный характеристический признак i -й категории.

Кроме того, можно заменить категорию числом входящих в неё объектов.

Существуют также "умные" способы кодирования когда категории кодируются какими-то интерпретируемыми значениями. Например, категорию товаров в интернет-магазине можно закодировать средней ценой товара. Тогда новый признак будет упорядочивать категории по дороговизне.

2 Метрики качества в задачах классификации и регрессии: accuracy, precision, recall, F1, ROC-AUC, log loss, MSE, MAE, квантильная, MAPE, SMAPE.

Когда какая из метрик предпочтительней. К оценке каких параметров распределений ответов приводят log loss, MSE и MAE (с обоснованием).

Ответ:

2.1 Метрики качества в задачах классификации.

2.1.1 Accuracy

Accuracy — доля правильных ответов при классификации.

$$\text{Accuracy} = \frac{\text{true answers}}{\text{all answers}} = \frac{T}{T + F}.$$

Accuracy бесполезна в задачах с неравными классами.

Пример: Допустим, мы хотим оценить работу спам-фильтра почты. У нас есть 100 не-спам писем, 90 из которых наш классификатор определил верно (True Negative = 90, False Positive = 10), и 10 спам-писем, 5 из которых классификатор также определил верно (True Positive = 5, False Negative = 5). Тогда: $\text{accuracy} = \frac{5+90}{5+90+10+5} = 86.4$.

Однако если мы просто будем предсказывать все письма как не-спам, то получим более высокую accuracy: $\frac{0+100}{0+100+0+10} = 90.9$.

2.1.2 Precision

Precision — точность (классификация на 2 класса). (количество сбитых самолётов / общее количество выстрелов).

Количество правильных положительных ответов / общее количество положительных ответов.

TP - истинно положительные

FP - ложно положительные

FN - ложно отрицательные

TN - истинно отрицательные

$$\text{precision} = \frac{\text{true positives}}{\text{all positives}} = \frac{TP}{TP + FP}.$$

2.1.3 Recall

Recall - полнота. (количество сбитых самолётов / общее количество самолётов). Количество правильных положительных ответов / каким должно быть количество положительных ответов.

$$\text{recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}} = \frac{TP}{TP + FN}.$$

Precision и recall не зависят, в отличие от accuracy, от соотношения классов и потому применимы в условиях несбалансированных выборок.

2.1.4 F-мера

F-мера (F-measure, F-score, F1) - среднее гармоническое между precision и recall. Значение F-measure ближе к меньшему из precision и recall.

$$F1 = 2 \frac{1}{\frac{1}{\text{recall}} + \frac{1}{\text{precision}}} = 2 \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}.$$

F-мера достигает максимума при полноте и точности, равными единице, и близка к нулю, если один из аргументов близок к нулю.

2.1.5 ROC-AUC

ЕСЛИ алгоритм бинарной классификации выдает не 0 и 1, а „вероятность 1“, то чтобы дать конкретные ответы, нужно выбрать порог. Чтобы оценивать алгоритм независимо от этого порога есть roc-auc. В нем перебирается этот порог, и для каждого порога на графике откладываются точки (FPR, TPR). Получится ROC-кривая. Теперь чтобы получить число, считается площадь под графиком AUC (area under ROC curve).

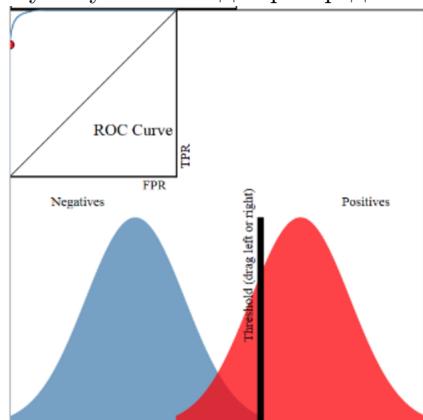
ROC-кривая - зависимость TPR (True Positive Rate) от FPR (False Positive Rate).

$$TPR = \frac{TP}{TP + FN}.$$

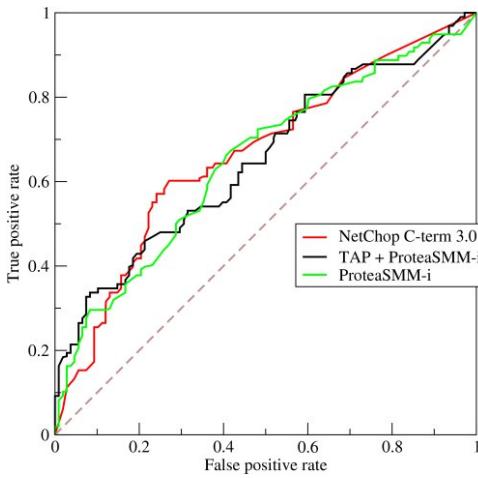
$$FPR = \frac{FP}{FP + TN}.$$

Как это делают?

Пусть у нас есть два распределения:



Мы просто перебираем границу и начинаем считать FPR и TPR, очевидны точки (1, 1) и (0,0). А дальше вычисляем эмпирически. Получаются вот такие графики:



2.2 Метрики качества в задачах регрессии.

2.2.1 LogLoss

Логарифмическая ошибка. Хорошо оценивает вероятность.

$$\text{LogLoss} = - \sum_{i=1}^l (y_i \ln p_i + (1 - y_i) \ln(1 - p_i)),$$

где y_i - реальная метка, p_i - предсказанная вероятность.

Если попросят доказать, помните, что $\text{LogLoss} = -\log P$, где P - правдоподобие.

2.2.2 MSE

MSE - mean squared error. Среднеквадратичное отклонение прогноза от исходного значения. Сильнее штрафует за большие по модулю отклонения.

$$MSE = \frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2,$$

где \hat{Y} - предсказанный результат, Y реальный.

2.2.3 MAE

MAE - mean absolute error. Отклонение прогноза от исходного значение, усреднённое по всем наблюдениям.

$$MAE = \frac{1}{n} \sum_{i=1}^n |\hat{Y}_i - Y_i|.$$

Среднеквадратичный функционал сильнее штрафует за большие отклонения по сравнению со среднеабсолютным, и поэтому более чувствителен к выбросам.

Среднеквадратичная ошибка подходит для сравнения двух моделей или для контроля качества во время обучения, но не позволяет сделать выводов о том, насколько хорошо данная модель решает задачу. Например, MSE = 10 является очень плохим показателем, если целевая переменная принимает значения от 0 до 1, и очень хорошим, если целевая переменная лежит интервале (10000, 100000).

Квантильная метрика (используется, например, тогда, когда заниженные предсказания опаснее завышенных).

$$Q(a, X^l) = \sum_{i=1}^l \rho_\tau(y_i - a(x_i)),$$

где $\rho_\tau(z) = (\tau - 1)[z < 0]z + \tau[z \geq 0]z$, $\tau \in [0, 1]$.

2.2.4 MAPE

MAPE - ошибка прогнозирования. Оценивается в процентах.

Расшифровывается Mean Average Percentage Error.

$$MAPE = \frac{100\%}{n} \sum_{i=1}^n \frac{|A_i - F_i|}{A_i},$$

где A - actual value, F - forecast.

2.2.5 SMAPE

SMAPE - ошибка прогнозирования. Оценивается в процентах.

Расшифровывается Symmetric Mean Absolute Percentage Error.

$$SMAPE = \frac{100\%}{n} \sum_{i=1}^n \left| \frac{A_i - F_i}{(|F_i| + |A_i|)/2} \right|$$

Напиши, когда какая предпочтительнее?

- 3 Решающие деревья. Как работает уже построенное решающее дерево. Рекурсивное построение деревьев. Энтропийный критерий и критерий gini. Классификационные и регрессионные деревья: в чем различия. Настройка гиперпараметров решающего дерева. Преимущества и недостатки деревьев.

3.1 Работа решающего дерева

Работает решающее дерево следующим образом:

Для одного конкретно взятого объекта последовательно в каждом узле проверяется соответствующий признак, в зависимости от результата происходит переход в какое-либо поддерево. Ответ получается в листе дерева.

Различия решающих деревьев в задаче классификации и задаче регрессии:

Регрессионное дерево, в отличие от классификационного отвечает в листьях не true/false, а каким-то действительным числом.

3.2 Как строятся решающие деревья?

- Строим разбиение выборки по значению одного из признаков. ("вот этот признак меньше какого-то порога"). Признак и порог нужно выбрать "наилучшим образом".
- Выборка делится по этому условию на две части.
- В каждой из них тоже можно сделать наилучшее разбиение.
- Процесс можно продолжать в тех узлов, в которые попадает достаточно много объектов. (можно ограничивать также глубину дерева и т.п.)

3.3 Как выдавать ответ в листе?

- Для задачи регрессии, если функционал – среднеквадратичная ошибка, то $a_m = \frac{1}{|X_m|} \sum_{i \in X_m} y_i$ - т.е. среднее значение в листе.
- В задаче классификации оптимально возвращать тот класс, который наиболее популярен среди объектов в X_m : $a_m = \operatorname{argmax}_{y \in Y} \sum_{i \in X_m} [y_i = y]$
- Если требуется указать вероятности классов, их можно указать как долю объектов разных классов в X_m : $a_{mk} = \frac{1}{|X_m|} \sum_{i \in X_m} [y_i = k]$

3.4 Как выбрать наилучшее разбиение?

Пусть Q - элементы, которые пришли в узел, L - элементы, которые ушли в левое поддерево, R - в правое.

$$G(j, t) = \frac{|L|}{|Q|} \cdot H(L) + \frac{|R|}{|Q|} \cdot H(R) \rightarrow \min_{j,t},$$

где j - индекс признака, t - ограничение на значение признака.

Примеры $H(R)$ (мера "неоднородности" множества R):

1. Энтропийный критерий

$$H(R) = -\sum_{k=1}^K p_k \ln p_k.$$

В этом выражении полагается, что $0 \ln 0 = 0$. Энтропийный критерий, как и критерий Джини, неотрицателен, а его оптимум также достигается только в том случае, когда все объекты в X относятся к одному классу. Энтропийный критерий имеет интересный физический смысл. Он заключается в том, что показывает, насколько распределение классов в X отличается от вырожденного. Энтропия в случае вырожденного распределения равна 0: такое распределение характеризуется минимальной возможной степенью неожиданности. Напротив, равномерное распределение самое неожиданное, и ему соответствует максимальная энтропия.

2. Критерий Gini

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

Все слагаемые в сумме неотрицательные, поэтому критерий Джини также неотрицателен. Его оптимум достигается только в том случае, когда все объекты в X относятся к одному классу. Одна из интерпретаций критерий Джини — это вероятность ошибки случайного классификатора. Классификатор устроен таким образом, что вероятность выдать класс k равна p_k .

(решаем задачу классификации на K классов, p_1, \dots, p_K доли объектов соответствующих классов в R)

Наилучшим разбиением становится такое, в котором в левом и правом поддереве какой-то класс доминирует. Для решения задачи регрессии достаточно взять среднеквадратичную ошибку в качестве $H(R)$:

$$H(R) = \frac{1}{|R|} \sum_{x_i \in R} (y_i - \bar{y})^2.$$

3.5 Настройка гиперпараметров

1. Выбор порога критерия останова, чтобы вершина стала листовой (например, если в вершину попало $< n$ объектов, то лист)
2. Подбор глубины дерева.

3.6 Преимущества и недостатки деревьев.

Достоинства:

1. Легко восстанавливают сложные зависимости
2. Просты для понимания и интерпретации
3. Не требуют нормализации и других способов предобработки данных
4. Могут работать одновременно с разными типами признаков (категориальные, числовые и пр.)
5. Можно оценить надёжность статистическими методами (подробнее)

Недостатки:

1. В каждом узле жадно выбирается оптимальное решение - слишком сильная эвристика, не может обеспечить оптимальность дерева в целом
2. Легко переобучаются
3. Если сравнительно немного изменить обучающую выборку, дерево может перестроиться очень сильно - нестабильность, которая используется в Random Forest.

4 Общие методы построения композиций - блэндинг, стэкинг, бэггинг и бустинг. Bias-variance trade-off (без вывода). Анализ бустинга и бэггинга с помощью bias-variance trade-off.

4.1 Bagging = Bootstrap aggregation

По схеме выбора с возвращением генерируем К обучающих выборок одинакового размера. Обучаем на них модель, усредняем результат (на семинаре говорилось, что для задачи классификации лучше брать к порядка \sqrt{d} , а для регрессии порядка $\frac{d}{3}$, где d - количество признаков). Важно, что в нагенеренных выборках сохраняются свойства большой выборки. Это семпл бэггинг.

Есть feature bagging. У нас есть фиксированная выборка, строим модель на рандомных подмножествах признаков.

4.2 Stacking

В обучающей выборке выделяем часть А, которая будет служить обучающей выборкой для M базовых алгоритмов. Оставшаяся часть обучающей выборки - часть В. На ней обучаем модель, комбинирующую базовые. (Считаем прогнозы базовых алгоритмов на выборке В. Эти прогнозы - новая матрица признаков. На них и обучаем новую более сложную модель, например, линейную регрессию.)

4.3 Blending

Строится несколько (небольшое количество) сильных алгоритмов. Для них подбираются веса, с которыми усредняются ответы. (веса в сумме должны давать 1)

Boosting

Жадное построение взвешенной суммы базовых алгоритмов $h_k(x)$. (В градиентном бустинге каждый следующий алгоритм обучается на ошибку предыдущего) $h_k(x)$, как правило, решающие деревья небольшой глубины или линейные модели.

4.4 Bias-variance trade-off

Если $L(x, D) = \text{Learn}(x, D)$ $T(x) = \text{Truth}(x)$

$$(L(x, D) - T(x))^2 = \text{Noise}^2 + \text{Bias}^2 + \text{Variance}$$

$\text{Noise}^2 = \text{lower bound on performance}$

$\text{Bias}^2 = (\text{expected error due to model mismatch})^2$

$\text{Variance} = \text{variation due to train sample and randomization}$

($bias^2 + variance$) то, что нужно посчитать, для предсказания. Часто возникают ситуации:

- low bias \Rightarrow high variance
- low variance \Rightarrow high bias

Tradeoff - проблема сравнения $bias^2$ vs. $variance$. Хотим уменьшить их сумму, уменьшить их одновременно не получится. В бэггинге $variance$ уменьшается (в нескоррелированном случае уменьшается в n раз, где n - число алгоритмов в композиции), а $bais$ не изменится. В бустинге мы уменьшаем $bais$, но из-за подстройки под данные увеличивается $variance$. Из-за этого можем переобучаться.

Если вдруг понадобится, то **Bias-variance decomposition**:

$$E_{x,y} E_{X^l} [(y - a_{X^l}(x))^2] = E_{x,y} E_{X^l} [(y - E_{X^l} a_{X^l}(x) + E_{X^l} a_{X^l}(x) - a_{X^l})^2] = E_{X^l} a_{X^l} [(y - E_{X^l} a_{X^l}(x))] + 2 \cdot E_{x,y} E_{X^l} [(y - E_{X^l} a_{X^l}(x))] + E_{x,y} E_{X^l} [(E_{x,y} E_{X^l} (x) - a_{X^l}(x))^2]$$

$$E_{x,y} E_{X^l} [(E_{x,y} E_{X^l} (x) - a_{X^l}(x))^2]$$

Посчитаем 2-ое слагаемое:

$$2 \cdot E_{x,y} E_{X^l} [(y - E_{X^l} a_{X^l}(x)) \cdot (E_{X^l} a_{X^l}(x) - a_{X^l}(x))] = 2 \cdot (E_{x,y} E_{X^l} [y \cdot E_{X^l} a_{X^l}(x) - E_{x,y} (E_{X^l} a_{X^l}(x))^2]) - E_{x,y} [(y \cdot E_{X^l} a_{X^l}(x) - E_{x,y} (E_{X^l} a_{X^l}(x))^2)]$$

0

Тогда:

$$E_{x,y} E_{X^l} [(y - a_{X^l}(x))^2] = E_{x,y} E_{X^l} [(y - E_{X^l} a_{X^l}(x))^2] + Var = E_{x,y} E_{X^l} [(y - E(y|x) + E(y|x) + a_{X^l}(x))^2] + Var = E_{x,y} [(y - E(y|x))^2] + E_{x,y} [(E(y|x) - E_{X^l} a_{X^l}(x))^2] + 2 \cdot E_{x,y} [(y - E(y|x))(E(y|x) - E_{X^l} a_{X^l}(x))] + Var$$

Теперь воспользуемся независимостью $E(E(y|x) - E_{X^l} a_{X^l}(x)) = 0$

$$2 \cdot E_{x,y} [(y - E(y|x)) \cdot (E(y|x) - E_{X^l} a_{X^l}(x))] = 2 \cdot E_{x,y} (y - E(y|x)) \cdot E_{x,y} (E(y|x) - E_{X^l} a_{X^l}(x)) = 0$$

То есть в конечном итоге мы получаем:

$$E_{x,y} E_{X^l} (y - a_{X^l}(x))^2 = \text{Noise} + \text{Bias}^2 + \text{Var}$$

$= ($

5 Бэггинг и Random Forest. Связь корреляции между ответами моделей и качеством модели в бэггинге.

Существует проблема корреляции деревьев в их композициях. Чем меньше корреляция, тем ближе алгоритм к идеальному:

$$(\text{Разброс композиции}) = (1/N)^*(\text{Разброс 1 базового алг-ма}) + (\text{Корреляция между базовыми алгоритмами})$$

Разброс — дисперсия ответов моделей, обученных по различным обучающим выборкам. Разброс характеризует то, насколько сильно прогноз алгоритма зависит от конкретной обучающей выборки.

Для уменьшения корреляции можно использовать например **бэггинг** (Обучение базовых алгоритмов происходит на случайных подвыборках обучающей выборки. Причем чем меньше размер случайной подвыборки, тем более независимы получаются базовые алгоритмы.)

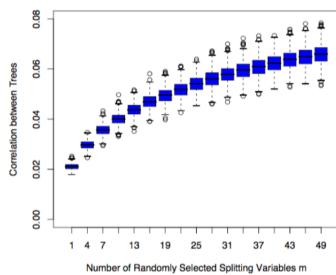
5.0.1 Бэггинг.

По схеме выбора с возвращением генерируем M обучающих выборок одинакового размера. Обучаем на них модель, усредняем результат.

5.0.2 Random Forest.

Рандомизировать процесс построения можно, если в задаче поиска оптимальных параметров выбирать j из случайногоподмножества признаков размера q . Оказывается, что этот подход действительно позволяет сделать деревья менее коррелированными.

По графику видно, что чем меньше «простор для выбора лучшего разбиения», то есть чем меньше q , тем меньше корреляции между получающимися решающими деревьями. Случай $q = 1$ соответствует абсолютно случайному выбору признака.



Для q есть некоторые рекомендации, которые неплохо работают на практике:

- В задаче классификации имеет смысл брать $q = \frac{d}{3}$, то есть использовать треть от общего числа признаков.
- В задаче классификации имеет смысл брать $q = \sqrt{d}$

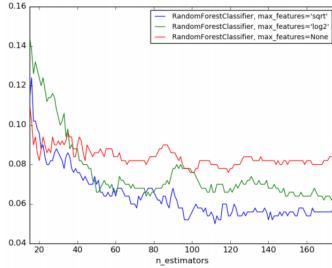
Построение Random Forest

Чтобы построить случайный лес из N решающих деревьев, необходимо:

1. Построить с помощью бутстрата N случайных подвыборок \hat{X}_n , $n = 1, \dots, N$.
2. Каждая получившаяся подвыборка \hat{X}_n используется как обучающая выборка для построения соответствующего решающего дерева $b_n(x)$. Причем:
 - Дерево строится, пока в каждом листе окажется не более n_{min} объектов. Очень часто деревья строят до конца ($n_{min} = 1$), чтобы получить сложные и переобученные решающие деревья с низким смещением.
 - Процесс построения дерева рандомизирован: на этапе выбора оптимального признака, по которому будет происходить разбиение, он ищется не среди всего множества признаков, а среди случайногоподмножества размера q .
 - Следует обратить особое внимание, что случайногоподмножество размера q выбирается заново каждый раз, когда необходимо разбить очередную вершину.
3. Построенные деревья объединяются в композицию:
 - В задачах регрессии $a(x) = \frac{1}{N} \sum_{n=1}^N b_n(x)$

- В задачах классификации $a(x) = \text{sign} \frac{1}{N} \sum_{n=1}^N b_n(x)$

Одна из особенностей случайных лесов: они не переобучаются при росте числа базовых алгоритмов:



То есть лес - это бэггинг над рандомизированными деревьями.

Чем меньше корреляция между ответами разных моделей, тем лучше качество большой модели - бэггинга. (Но это при условии одинакового качества маленьких моделей)

Проблема Random Forest в том, что это композиция глубоких деревьев, которые строятся независимо друг от друга. Обучение глубоких деревьев требует очень много вычислительных ресурсов, особенно в случае большой выборки или большого числа признаков.

Одна из особенностей случайных лесов: они не переобучаются при росте числа базовых алгоритмов.

6 Бустинг и GBM. Выбор параметров в ансамблях решающих деревьев. Сравнение Random Forest и GBDT (подбор параметров, переобучение).

Сначала, что такое **бустинг**.

Бустинг - итерационный алгоритм, реализующий "сильный" классификатор, который позволяет добиться произвольно малой ошибки обучения (на обучающей выборке) на основе композиции "слабых" классификаторов, каждый из которых лучше, чем просто угадывание, т.е. вероятность правильной классификации больше 0.5.

Ключевая идея: использование весовой версии одних и тех же обучающих данных вместо случайного выбора их подмножества

В чем отличие от бэггинга пока не очень понятно. Уточним:

- Бэггинг - примеры выбираются так, что каждый пример имеет одинаковые шансы попасть в обучающую подвыборку.
- Бустинг.
 - Базовые алгоритмы строятся последовательно, один за другим.
 - Каждый следующий алгоритм строится таким образом, чтобы исправлять ошибки уже построенной композиции.

6.1 Gradient boosting

Gradient boosting - последовательность решающих деревьев, где каждое следующее дерево обучается на ошибку предыдущего. Как следствие, при большой глубине (длине последовательности) переобучается.

На практике глубина берется около 3-5. Количество деревьев сотни, а то и тысячи. (На одном из наших контестов идеально получалось 4 и 150). Количество деревьев ограничивается для того, чтобы не допустить переобучения.

В random forest (как и в gb) глубина обычно выбирается не очень большой. Можно вообще определять ее автоматически, вводя пороговое значение для функции критерия (энтропийного, gini, ...).

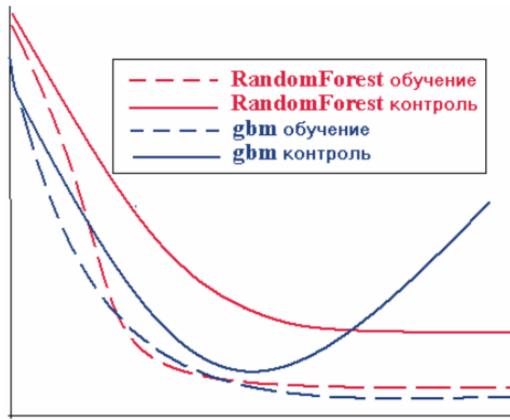
В random forest переобучения добиться довольно сложно, поэтому количество деревьев ограничивается из соображений времени работы. Кроме того при большом количестве деревьев начинают появляться похожие, а скоррелированные деревья пользы не приносят, так что делать их количество слишком бессмысленно.

Сравнение:

- RF почти не требует настройки
- GB в среднем работает лучше
- RF можно параллелизировать, GB определенно последовательный

Важный момент, который обсуждался на семинаре 494 группы: при решении задачи регрессии, если в test все ответы были неотрицательными, то на train RF будет давать неотрицательные ответы, чего нельзя сказать о GB!

Бэггинг и бустинг: переобучение



6.2 Обучение базовых алгоритмов в GB

Обучение базовых алгоритмов происходит последовательно. Пусть к некоторому моменту обучены $N - 1$ алгоритмов $b_1(x), \dots, b_{N-1}(x)$, то есть композиция имеет вид:

$$a_{N-1}(x) = \sum_{n=1}^{N-1} b_n(x).$$

Теперь к текущей композиции добавляется еще один алгоритм $b_N(x)$. Этот алгоритм обучается так, чтобы как можно сильнее уменьшить ошибку композиции на обучающей выборке:

$$\sum_{i=1}^l L(y_i, a_{N-1}(x_i) + b(x_i)) \rightarrow \min_b.$$

Сначала имеет смысл решить более простую задачу: определить, какие значения s_1, \dots, s_l должен принимать алгоритм $b_N(x_i) = s_i$ на объектах обучающей выборки, чтобы ошибка на обучающей выборке была минимальной:

$$F(s) = \sum_{i=1}^l L(y_i, a_{N-1}(x_i) + s_i) \rightarrow \min_s,$$

где $s = (s_1, \dots, s_l)$ - вектор сдвигов.

Другими словами, необходимо найти такой вектор сдвигов s , который будет минимизировать функцию $F(s)$. Поскольку направление наискорейшего убывания функции задается направлением антиградиента, его можно принять в качестве вектора s :

$$s = -\nabla(F) = \begin{pmatrix} -L'_z(y_1, a_{N-1}(x_1)), \\ \vdots \\ -L'_z(y_l, a_{N-1}(x_l)) \end{pmatrix}. \quad (1)$$

Компоненты вектора сдвигов s , фактически, являются теми значениями, которые на объектах обучающей выборки должны принимать новый алгоритм $b_N(x)$, чтобы минимизировать ошибку строящейся композиции.

Обучение $b_N(x)$, таким образом, представляет собой задачу обучения на размеченных данных, в которой $\{(x_i, s_i)\}_{i=1}^l$ - обучающая выборка, и используется, например, квадратичная функция ошибки:

$$b_N(x) = \operatorname{argmin}_b \frac{1}{l} \sum_{i=1}^l (b(x_i) - s_i)^2.$$

Следует обратить особое внимание на то, что информация об исходной функции потерь $L(y, z)$, которая не обязательно является квадратичной, содержится в выражении для вектора оптимального сдвига s . Поэтому для большинства задач при обучении $b_N(x)$ можно использовать квадратичную функцию потерь.

6.3 Описание алгоритма градиентного бустинга

1. **Инициализация:** инициализация композиции $a_0(x) = b_0(x)$, то есть построение простого алгоритма b_0 .

2. **Шаг итерации:**

(a) Вычисляется вектор сдвига

$$s = -\nabla(F) = \begin{pmatrix} -L'_z(y_1, a_{N-1}(x_1)), \\ \vdots \\ -L'_z(y_l, a_{N-1}(x_l)) \end{pmatrix}. \quad (2)$$

(b) Строится алгоритм

$$b_N(x) = \operatorname{argmin}_b \frac{1}{l} \sum_{i=1}^l (b(x_i) - s_i)^2.$$

параметры которого подбираются таким образом, что его значения на элементах обучающей выборки были как можно ближе к вычисленному вектору оптимального сдвига s .

(c) Алгоритм $b_n(x)$ добавляется в композицию

$$a_n(x) = \sum_{m=1}^n b_m(x).$$

3. Если не выполнен критерий останова (об этом будет рассказано далее), то выполнить еще один шаг итерации. Если критерий останова выполнен, остановить итерационный процесс.

6.4 Градиентный бустинг для регрессии

Типичный функционал ошибки в регрессии — это среднеквадратичная ошибка:

$$MSE(a, X) = \frac{1}{l} \sum_{i=1}^l (a(x_i) - y_i)^2.$$

При этом функция потерь, которая измеряет ошибку для одного объекта:

$$L(y, z) = (z - y)^2, \quad L'_z(y, z) = 2(z - y),$$

где z — это прогноз нашего алгоритма, а y — истинный ответ на данном объекте. Соответственно, вектор сдвигов, каждая компонента которого показывает, как нужно модифицировать ответ на каждом объекте обучающей выборки, чтобы уменьшить среднеквадратичную ошибку, имеет вид:

$$s = \begin{pmatrix} -2(a_{N-1}(x_1) - y_1) \\ \vdots \\ -2(a_{N-1}(x_l) - y_l) \end{pmatrix}. \quad (3)$$

6.5 Градиентный бустинг для классификации

В задаче бинарной классификации ($\mathbb{Y} = -1, +1$) популярным выбором для функции потерь является логистическая функция потерь:

$$\sum_{i=1}^n \log(1 + \exp(-y_i a(x_i))),$$

где $a(x) \in \mathbb{R}$ — оценка принадлежности положительному классу. Если $a(x) > 0$, классификатор относит объект x к классу $+1$, а при $a(x) \leq 0$ — к классу -1 . Причем, чем больше $|a(x)|$, тем увереннее классификатор в своем выборе. Функция потерь в этом случае записывается следующим образом:

$$L(y, z) = \log(1 + \exp(-yz)), \quad L'_z(y, z) = -\frac{y}{1 + \exp(yz)}.$$

Вектор сдвигов s в этом случае будет иметь вид:

$$s = \begin{pmatrix} \frac{y_1}{1+exp(y_1 a_{N-1}(x_1))} \\ \vdots \\ \frac{y_l}{1+exp(y_l a_{N-1}(x_l))} \end{pmatrix}. \quad (4)$$

Новый базовый алгоритм будет настраиваться таким образом, чтобы вектор его ответов на объектах обучающей выборки был как можно ближе к s . После того, как вычислен алгоритм $a_N(x)$, можно оценить вероятности принадлежности объекта x к каждому из классов:

$$P(y = 1 | x) = \frac{1}{1 + exp(-a_N(x))}, \quad P(y = -1 | x) = \frac{1}{1 + exp(a_N(x))}.$$

6.6 Градиентный бустинг для решающих деревьев

В градиентном бустинге каждый новый базовый алгоритм b_N прибавляется к уже построенной композиции:

$$a_N(x) = a_{N-1}(x) + b_N(x).$$

Если базовые алгоритмы - это решающие деревья

$$b_N(x) = \sum_{j=1}^J [x \in R_{N_j}] b_{N_j},$$

тогда новая композиция a_N будет выглядеть следующим образом:

$$a_N(x) = a_{N-1}(x) + \sum_{j=1}^J [x \in R_{N_j}] b_{N_j},$$

где R_i , $i = 1,..J$ – области, на которые дерево разбивает пространство, а b_j – предсказание дерева на области R_j (напомним, что дерево на какой-то области отвечает константой). $[x \in R_j]$ – индикатор того, что объект x попал в область R_j .

Последнее выражение можно проинтерпретировать не только как прибавление одного решающего дерева, но и как прибавление J очень простых алгоритмов, каждый из которых возвращает постоянное значение в некоторой области и ноль во всем остальном пространстве. Можно подобрать каждый прогноз b_{N_j} , где N - номер дерева, j - номер листа в этом дереве, таким образом, чтобы он был оптимальным с точки зрения исходной функции потерь:

$$\sum_{i=1}^l L(y_i, a_{N-1}(x) + \sum_{j=1}^J [x \in R_{N_j}] b_{N_j}) \rightarrow \min_{b_1, \dots, b_J}.$$

Можно показать, что данная задача распадается на J подзадач:

$$b_{N_j} = argmin_{\gamma \in R} \sum_{x_i \in R_j} L(y_i, a_{N-1}(x) + \gamma).$$

Такая задача часто решается аналитически или любым простым методом. Итак, структура базового решающего дерева (структура областей R_j) в градиентном бустинге настраивается минимизацией среднеквадратичной ошибки. Потом можно переподобрать ответы в листьях, то есть перенастроить их, так, чтобы они были оптимальны не с точки зрения среднеквадратичной ошибки (с помощью которой строилось дерево), а с точки зрения исходной функции потерь L . Это позволяет существенно увеличить скорость сходимости градиентного бустинга.

Например, в задаче классификации с логистической функцией потерь, практически оптимальные значения для прогнозов в листьях имеют вид:

$$b_{N_j} = -\frac{\sum_{x_i \in R_j} s_i}{\sum_{x_i \in R_j} |s_i|(1 - |s_i|)}.$$

7 Линейные модели в задачах классификации и регрессии: функции потерь, регуляризаторы, оптимизационные задачи. Стохастический градиентный спуск.

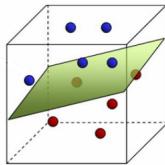
Линейный классификатор:

Формализуем линейный классификатор

$$a(x) = \begin{cases} 1, & \text{если } f(x) > 0 \\ -1, & \text{если } f(x) < 0 \end{cases}$$

$$f(x) = w_0 + w_1x_1 + \dots + w_nx_n = w_0 + \langle w, x \rangle$$

Геометрическая интерпретация:
разделяем классы плоскостью



Отступом алгоритма $a(x) = sign\{f(x)\}$ на объекте x_i называется величина $M_i = y_i f(x_i)$ (y_i - класс, к которому относится объект x_i).

$$\begin{aligned} M_i \leq 0 &\Leftrightarrow y_i \neq a(x_i) \\ M_i > 0 &\Leftrightarrow y_i = a(x_i) \end{aligned}$$

Функция потерь

$$Q(w) = \sum_{i=1}^{\ell} [M_i(w) < 0] \leq \tilde{Q}(w) = \sum_{i=1}^{\ell} \mathcal{L}(M_i(w)) \rightarrow \min_w$$

Функция эмпирического риска
Функция потерь

Примеры функций потерь:

$$Q(M) = (1 - M)^2$$

$$V(M) = (1 - M)_+$$

$$S(M) = 2(1 + e^M)^{-1}$$

$$L(M) = \log_2(1 + e^{-M})$$

$$E(M) = e^{-M}$$

Регуляризаторы.

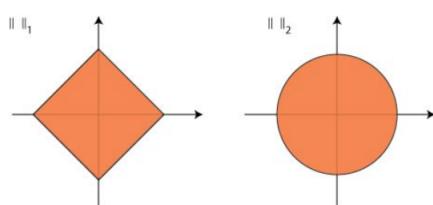
Идея регуляризаторов - добавление ограничений на коэффициенты (их излишний рост приводит к переобучению).

Регуляризация

$$\sum_{i=1}^l L(M_i) + \gamma \sum_{k=1}^m |w_k| \rightarrow \min \quad \sum_{i=1}^l L(M_i) + \gamma \sum_{k=1}^m w_k^2 \rightarrow \min$$

l1 – регуляризация

l2 – регуляризация

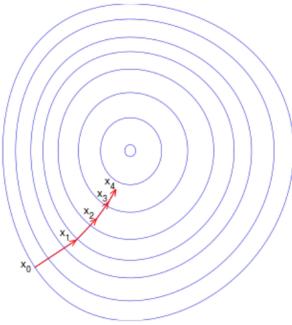


Стохастический градиентный спуск.

Суть градиентного спуска - минимизация функции с помощью последовательных небольших шагов в сторону антиградиента (в сторону убывания функции).

Градиентный спуск

$$\mathbf{x}_{n+1} = \mathbf{x}_n - \gamma_n \nabla F(\mathbf{x}_n), \quad n \geq 0.$$



$$\nabla_w \tilde{Q} = \sum_{i=1}^l \nabla L(M_i)$$

$$\nabla \tilde{Q} = \sum_{i=1}^l L'(M_i) \frac{\partial M_i}{\partial w}$$

$$\frac{\partial M_i}{\partial w} = y_i x_i$$

$$\nabla \tilde{Q} = \sum_{i=1}^l y_i x_i L'(M_i)$$

$$w_{n+1} = w_n - \gamma_n \sum_{i=1}^l y_i x_i L'(M_i)$$

Стохастический градиент

$$w_{n+1} = w_n - \gamma_n \sum_{i=1}^l y_i x_i L'(M_i)$$

$$w_{n+1} = w_n - \gamma_n y_i x_i L'(M_i)$$

x_i – случайный элемент обучающей выборки

Если я правильно понимаю, γ_n - это шаг градиентного спуска. (некоторое небольшое положительное число)

8 Линейная регрессия. Геометрический и аналитический вывод формулы для весов признаков. Регуляризация в линейной регрессии: гребневая регрессия и LASSO.

Постановка задачи.

Линейная регрессия

$$a(x) = \langle w, x \rangle + w_0$$

$$Q = \sum_{i=1}^N L(y_i, a(x_i))$$

$$L(y_i, a(x_i)) = (y_i - a(x_i))^2$$

$$L(y_i, a(x_i)) = |y_i - a(x_i)|$$

Здесь $a(x)$ - ответ алгоритма на элементе x , Q - функционал эмпирического риска, L - функция потерь.

Модель и матричная запись

$$\begin{pmatrix} y_1 \\ y_2 \\ \dots \\ y_l \end{pmatrix} \approx \begin{pmatrix} \widehat{y}_1 \\ \widehat{y}_2 \\ \dots \\ \widehat{y}_l \end{pmatrix} = \begin{pmatrix} x_1^T \\ x_2^T \\ \dots \\ x_l^T \end{pmatrix} w$$

\uparrow \uparrow \uparrow
 $y \approx \widehat{y} = Fw$
 $w = \underset{w}{\operatorname{argmin}} \|y - \widehat{y}\|^2$

y - реальные ответы, \widehat{y} - ответы алгоритма, F - матрица признаков, w - веса признаков.
Выход формулы для весов признаков.

Аналитический вывод

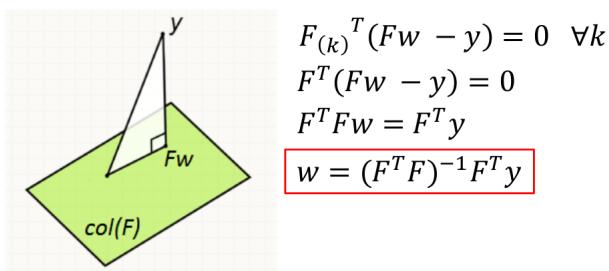
$$\frac{\partial(y - Fw)^2}{\partial w} = 2F^T(y - Fw) = 0$$

$$F^T F w = F^T y$$

$$w = (F^T F)^{-1} F^T y$$

Геометрическая интерпретация

$$(Fw - y) \perp F_{(k)} \quad \forall k = 1, \dots, m$$



Кусочек из другого места, который помогает понять геометрическую интерпретацию:

Оказывается, у решения МНК существует наглядная геометрическая интерпретация. Заметим, что $A\alpha \in \mathbb{R}^\ell$ – линейная комбинация столбцов матрицы A . Обозначим с помощью $\text{Lin}(A)$ линейное подпространство, натянутое на столбцы матрицы A .

Задача. Найдите α^* , для которого $A\alpha^*$ – ортогональная проекция вектора Y на подпространство $\text{Lin}(A)$.

Мы хотим найти такой вектор α^* , для которого вектор $Y - A\alpha^*$ ортогонален всем векторам подпространства $\text{Lin}(A)$.

Регуляризация в линейной регрессии.

LASSO - l1-регуляризация для линейной классификации, гребневая регрессия (Ridge) - l2-регуляризация.

Гребневая регрессия (ℓ_2 -регуляризация)

$$\sum_{i=1}^l (a(x_i) - y_i)^2 + \gamma \sum_{k=1}^m w_k^2 \rightarrow \min$$

$$\frac{\partial((y - Fw)^2 + \gamma w^2)}{\partial w} = 2F^T(y - Fw) + 2\gamma w = 0$$

$$(F^T F + \gamma I)w = F^T y$$

$$w = (F^T F + \gamma I)^{-1} F^T y$$

На слайде ошибка допущена при дифференцировании: потерян – перед первым слагаемым. Итоговая формула верна. Одна из проблем классической линейной регрессии – плохая обусловленность матрицы $F^T F$, из-за чего не получается хорошо обращать её. Бороться с этим можно по-разному и один из способов – гребневая регрессия. Применяя её, мы к матрице $F^T F$ добавляем γ на диагонали, за счет чего увеличиваем все собственные числа, а значит и улучшаем обусловленность. Так же есть теорема, что существует γ такое, что $\mathbb{E}(\hat{w}_{ridge} - w)^2 < \mathbb{E}(\hat{w}_{common} - w)^2$, где w – "настоящий" вектор весов (здесь подразумевается, что действительно имеет место линейная зависимость). Но как искать такое γ на практике не придумали.

9 Логистическая регрессия. Варианты записи оптимизационной задачи. Оценка вероятности принадлежности к классу. Настройка параметров с помощью стохастического градиентного спуска.

Логистическая регрессия – задача линейной оптимизации с функцией потерь $L(M) = \log(1 + e^{-M})$

Логистическая регрессия

$$y_i \in \{0, 1\} \quad Q = - \sum_{i=1}^{\ell} y_i \ln p_i + (1 - y_i) \ln(1 - p_i) \rightarrow \min_w$$

$$p_i = \sigma(\langle w, x_i \rangle) = \frac{1}{1 + e^{-\langle w, x_i \rangle}} = P(y = 1|x)$$

Как правило, добавляется ℓ_1 или ℓ_2 -регуляризация, а оптимизационная задача решается с помощью SGD или метода Ньютона-Рафсона

торые имеют сразу несколько интересных последствий. Во-первых, линейный алгоритм классификации оказывается оптимальным байесовским классификатором. Во-вторых, однозначно определяется функция потерь. В-третьих, возникает интересная дополнительная возможность наряду с классификацией объекта получать численные оценки вероятности его принадлежности каждому из классов.

Оценка вероятности принадлежности к классу.

$$P(y=1|x) = \frac{1}{1+e^{-\langle w, x_i \rangle}}$$

- это предположение.

$$\text{Тогда получаем, что } P(y=0|x) = 1 - P(y=1|x) = 1 - \frac{1}{1+e^{-\langle w, x_i \rangle}}$$

$$\text{Обозначим } f(x) = \frac{1}{1+e^{-x}}$$

Тогда для краткости функцию распределения y при заданном x можно записать в виде:

$$P(y|x) = f(\langle w, x \rangle)^y \cdot (1 - f(\langle w, x \rangle))^{1-y}, \text{ здесь } y \in \{0, 1\}$$

Фактически, это распределение бернулли с параметром $f(\langle w, x \rangle)$.

Варианты записи оптимизационной задачи.

Можно рассмотреть две оптимизационные задачи.

Первая для классов $y_i \in \{0, 1\}$.

Нужно минимизировать функционал риска (см слайд)

Эквивалентность оптимизационных задач

$$Q = - \sum_{i=1}^{\ell} y_i \ln \frac{1}{1 + e^{-\langle w, x_i \rangle}} + (1 - y_i) \ln \frac{1}{1 + e^{\langle w, x_i \rangle}} \rightarrow \min_w$$

$$-y_i \ln \frac{1}{1 + e^{-\langle w, x_i \rangle}} - (1 - y_i) \ln \frac{1}{1 + e^{\langle w, x_i \rangle}} = \begin{cases} \ln(1 + e^{-\langle w, x_i \rangle}), & y_i = 1 \\ \ln(1 + e^{\langle w, x_i \rangle}), & y_i = 0 \end{cases}$$

$$Q = \sum_{i=1}^{\ell} \underbrace{\ln(1 + e^{-y_i \langle w, x_i \rangle})}_{L(M)} \rightarrow \min_w \quad y_i \in \{-1, 1\}$$

Таким образом получаем эквивалентную исходной задачу, но уже для классов $y_i \in \{-1, 1\}$.

Настройка параметров с помощью стохастического градиентного спуска.

(см. билет 7)

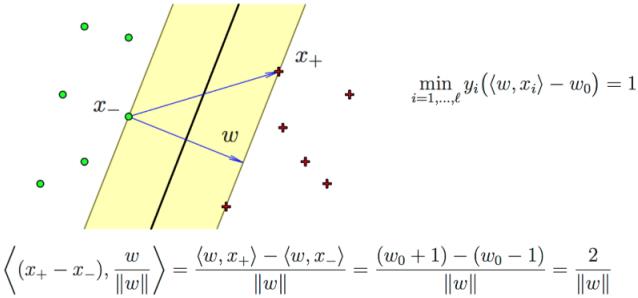
10 Метод опорных векторов: оптимизационная задача в условной и безусловной форме. Опорные векторы (достаточно записать и проанализировать Лагранжиан, оптимизационная задача, выраженная через двойственные переменные - опционально). Идея Kernel Trick.

Метод опорных векторов это по сути - линейный классификатор с кусочно-непрерывной функцией потерь hinge loss ($\max(0, 1 - M)$, M - margin $M = y_i(\langle w, x_i \rangle - w_0)$) и L_2 -регуляризатором.

Теперь о том, как был придуман этот метод (а он был придуман из других соображений, а не просто общего вида линейной регрессии).

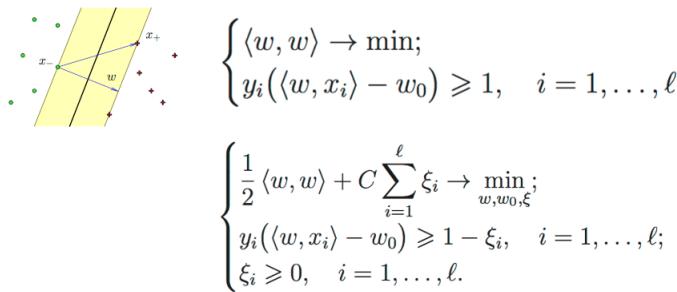
Итак, представим, что у нас есть выборка и нам надо разделить ее на 2 класса, и пусть она разделяется гиперплоскостью. (пока предполагаем, что она разделяется). Тогда пусть она разделяется гиперплоскостью $\langle w, x \rangle = w_0$. Надо выбрать лучшую плоскость из всех возможных! Как это сделать? Посмотреть на ширину полосы, которая разделяет 2 выборки. Для этого посмотрим на "крайние" элементы двух классов: x_+ , x_- . Так как домножение w и w_0 не влияет на гиперплоскость, то давайте предположим, что на "крайних элементах" уравнение гиперплоскости будет давать +1 и -1 соответственно. То есть формально: $\min_{y_i} y_i(\langle w, x_i \rangle - w_0) = 1$ (умножение на y_i , чтобы было +1, а не +1 и -1, минимум - тк условие на крайние элементы.) Теперь посмотрим, чему равна ширина полосы. Это просто проекция вектора $x_+ - x_-$ на w .

Ширина разделяющей полосы



Отлично, получается, чтобы максимизировать ширину полосы надо минимизировать норму весов. И не забыть про условие, что $y_i(\langle w, x_i \rangle - w_0) \geq 1$. А теперь давайте вспомним, что у нас данные обычно не разделяются идеально. Добавим ошибку - ξ_i . $y_i(\langle w, x_i \rangle - w_0) \geq 1 - \xi_i$ - добавление ошибки. Сумму добавляем, чтобы ошибки все-таки не были большими. А условие ≥ 0 , вроде, очевидно... Ура! Условная задача SVM!

Случай линейно неразделимой выборки



Теперь к безусловной. Возьмем и перепишем все уравнения на ξ_i в терминах M (margin). Понятно, что решением будет $\max(0, 1 - M)$. Всё, можно забыть на все условия на ошибки, кроме минимизации. Фух, вернемся к первому уравнению до переписывания в M. И поймем, что осталось только оно, то теперь переписанное в M - безусловная форма SVM!

Безусловная оптимизационная задача в SVM

$$\begin{cases} \frac{1}{2} \langle w, w \rangle + C \sum_{i=1}^{\ell} \xi_i \rightarrow \min_{w, w_0, \xi}; \\ y_i(\langle w, x_i \rangle - w_0) \geq 1 - \xi_i, \quad i = 1, \dots, \ell; \\ \xi_i \geq 0, \quad i = 1, \dots, \ell. \end{cases}$$

Напоминание:
 $M_i = y_i(\langle w, x_i \rangle - w_0)$
 отступ на i-том объекте

$$\begin{aligned} \xi_i &\geq 0 \\ \xi_i &\geq 1 - M_i \\ \sum_{i=1}^l \xi_i &\rightarrow \min \\ Q(w, w_0) &= \sum_{i=1}^{\ell} (1 - M_i(w, w_0))_+ + \frac{1}{2C} \|w\|^2 \rightarrow \min_{w, w_0} \end{aligned}$$

Опорные вектора: В общем, берем мы такую условную форму SVM и применяем теорему Каруша-Куна-Таккера. Ищем двойственную задачу, так сказать. Для этого надо написать тот самый Лагранжиан и его производную. Получится:

Опорные векторы в SVM

$$\begin{aligned}\mathcal{L}(w, w_0, \xi; \lambda, \eta) &= \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{\ell} \xi_i - \sum_{i=1}^{\ell} \lambda_i (M_i(w, w_0) - 1 + \xi_i) - \sum_{i=1}^{\ell} \xi_i \eta_i = \\ &= \frac{1}{2} \|w\|^2 - \sum_{i=1}^{\ell} \lambda_i (M_i(w, w_0) - 1) - \sum_{i=1}^{\ell} \xi_i (\lambda_i + \eta_i - C),\end{aligned}$$

$$\frac{\partial \mathcal{L}}{\partial w} = w - \sum_{i=1}^{\ell} \lambda_i y_i x_i = 0 \quad \Rightarrow \quad w = \sum_{i=1}^{\ell} \lambda_i y_i x_i;$$

Идея KernelTrick: иногда не разделяется гиперплоскостью, а разделить хотим - ну же сделаем преобразование координат. Ну а зачем нам делать его явно? Го сделаем неявно - зададим просто новое скалярное произведение. $K(w, x) = \langle \phi(x), \phi(w) \rangle$, ϕ - преобразование координат типа. Например, полиномиальное ядро: $K(w, x) = (a \langle w, x \rangle + b)^d$ Радиальное ядро: $K(w, x) = e^{-a|w-x|^2}$

Капитан функциональный анализ информирует: $K(w, x)$ не обязано быть скалярным произведением по всем аксиомам, как это видно на примерах выше. Просто подбирается функция из геометрического смысла.

11 Нейронные сети, обучение (backprop), слои для нейронных сетей (dense, conv, pooling, batchnorm), нелинейности (relu vs sigmoid, softmax), функции потерь (logloss, l2, hinge)

Нейронная сеть - вычислительная модель, содержащая большое количество простых элементов, которые способны обрабатывать информацию в зависимости от своего состояния.

Нейросети обычно состоят из трёх частей:

- Input Layer** Начальный слой, который служит для того, чтобы пропустить через сеть очередную порцию данных. У него нет параметров, количество нейронов в нём фиксировано и не является гиперпараметром.
- Hidden Layers** Обычно это несколько (в глубоких сетках сотни) различных слоёв, размер которых фиксирован архитектурой (это один из гиперпараметров), но значения параметров меняются в процессе обучения. (Хотя можно сделать и полностью фиксированный слой).
- Output Layer** Выходной слой, количество нейронов в котором фиксировано, не содержит параметров. Обычно используется для подсчета лосса.

11.1 Обучение нейронных сетей

Как обучаются нейронные сети?

Нейронная сеть обучается итеративно, берётся мини-батч данных и пропускается через сеть.

Обычно, мы используем алгоритм градиентного спуска, в нейросетях используются, в основном, его модификации, т.к. градиентный спуск сходится долго.

BackPropogation это метод вычисления градиента лосс-функции по параметрам.

[Лекция, разжевывающая backprop.](#)

Нейросеть - это некая (может быть, очень-очень большая и громоздкая) функция $f(X, W1, \dots)$. Она представима как композиция некоторых функций, поэтому можно построить график вычислений (обычно он строится топологической сортировкой по функциям).

Рассмотрим картинку:

Backpropagation: a simple example

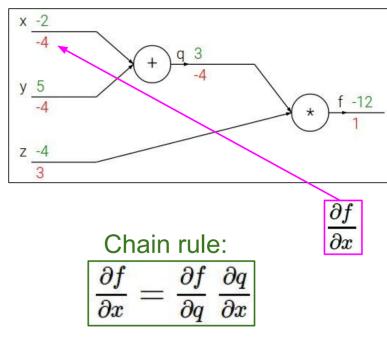
$$f(x, y, z) = (x + y)z$$

$$\text{e.g. } x = -2, y = 5, z = -4$$

$q = x + y$	$\frac{\partial q}{\partial x} = 1, \frac{\partial q}{\partial y} = 1$
-------------	--

$f = qz$	$\frac{\partial f}{\partial q} = z, \frac{\partial f}{\partial z} = q$
----------	--

Want: $\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, \frac{\partial f}{\partial z}$



На картинке у нас функция f , сначала она разбивается на граф вычислений, показанный на рисунке.

Пусть x, y, z – наши параметры, а $f(x, y, z)$ – значение нашей функции. Мы хотим узнать градиенты функции f по параметрам.

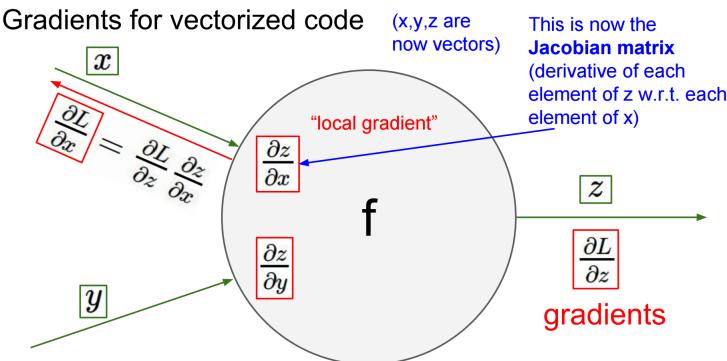
Рассмотрим самую правую вершинку, у нас есть 2 входа – $x + y$, который мы обозначим q , и z .

В начале мы всегда рассматриваем $\frac{\partial f}{\partial f}$, что всегда единица, дальше переходим к входам, т.е. z и $x + y$.

$\frac{\partial f}{\partial z} = x + y$ и вычисляется сразу, а вот с вершинкой $x + y$ сложнее. Пусть $q = x + y$, тогда для того, чтобы посчитать $\frac{\partial f}{\partial x}$ и $\frac{\partial f}{\partial y}$ нужно воспользоваться Chain rule (как на картинке). Для этого нам нужно знать $\frac{\partial f}{\partial q}$, что мы уже посчитали на предыдущем шаге и $\frac{\partial q}{\partial x}$, что мы можем посчитать без участия f .

Таким образом, мы посчитали производные по параметрам. Для этого нам нужны были некоторые производные для chain rule – производная по входу (например, $\frac{\partial q}{\partial x}$) и по выходу ($\frac{\partial f}{\partial q}$). Заметим, что первую производную мы можем посчитать, не зная ничего о выходах (т.е. более глубоких слоях сети). Поэтому в реальных сетях они считаются в процессе forward-pass и хранятся в вершинках графа вычислений. А вот вторую производную посчитать forward pass'ом мы никак не можем, вот где и нужен backward pass.

Рассмотрим картинку, объясняющую это явление



Тут показан шаг backprop'a для вычисления производной, причем сразу для векторных переменных.

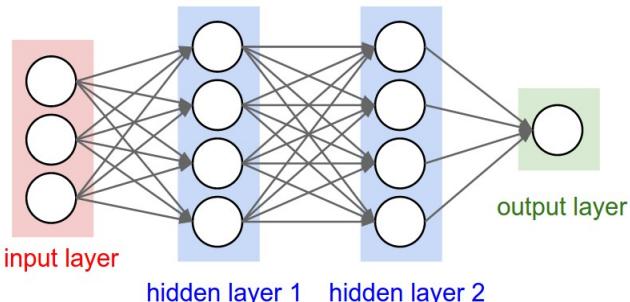
На фазе forward-pass (когда вы пропускаете данные через сеть, умножая на матрицы и прочее), вычисляется выход вершинки z , который определяется через какую-то функцию, затем вычисляются производные выхода по выходам, т.е. $\frac{\partial z}{\partial x}$ и $\frac{\partial z}{\partial y}$, т.к. все данные для этого у нас есть.

После фазы forward pass'a начинается backward pass, где мы просто применяем Chain Rule на уже вычисленных производных, вычисляя производные по всем параметрам. В этом-то умножении (chain rule) и состоит проблема затухания градиента. Если ваш градиент станет очень маленьким в каком-то месте, то это просто не даст другим градиентам быть не около 0, т.к. при умножении получается очень маленькое число. В этом случае gradient flow прекращается и сеть перестает учиться. Это немного решается с помощью shortcut'ов в ResNet.

Отсюда сразу вытекают последствия: для векторных операций $\frac{\partial z}{\partial x}$, вычисляемый на фазе forward pass'a, это Якобиан, а поэтому он очень большой. Поэтому мы зачастую не можем загрузить весь датасет в память – нам же нужно хранить производные, полученные forward pass'ом. Поэтому используются всякие хитрости, о которых будет идти речь ниже.

11.2 Слои для нейронных сетей

1. Dense:

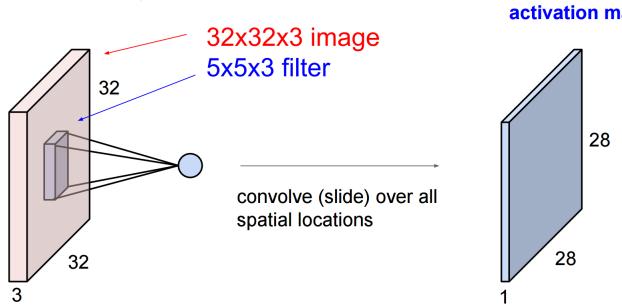


Dense Layer или Fully Connected Layer – слой, состоящий из n нейронов. Пусть предыдущий слой имел k выходов, тогда к каждому нейрону в FC слое идет k взвешенных рёбер, после этого (обычно) берётся их взвешенная сумма и применяется нелинейность.

Очевидно, что такую операцию можно представить умножением матрицы весов W на вектор-инпут x . Так как все тут (пока не говорим о нелинейностях) комбинация умножений и суммирований, то такая функция дифференцируема, а значит, можно использовать backprop для обучения сети, состоящей из таких слоёв.

2. Conv:

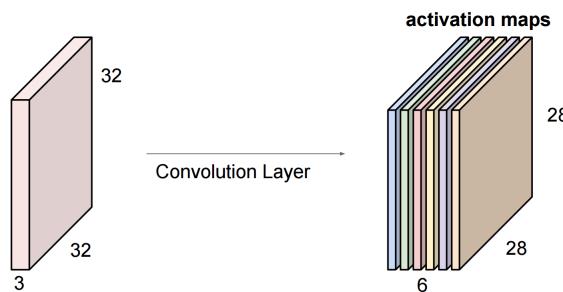
Convolution Layer



Convolutional Layer или сверточный слой. Идея в том, чтобы анализировать части наших данных независимо, поэтому вводится понятие фильтра. На картинке фильтр размера 5x5, третья размерность **обязательно** должна совпадать с глубиной (3-й размерностью) данных, если речь идет о 2D свертке (во всех библиотеках это делается автоматически, нужно лишь задать одну размерность h, и вы получите фильтр h x h). Между частями картинки и фильтром берётся скалярное произведение и прибавляется bias: $w^T \cdot x + b$. Таким образом получается одно число. Если пройтись таким фильтром последовательно по всей картинке, получится Activation Map.

Применив много различных фильтров, вы получите кучу Activation Maps:

For example, if we had 6 5x5 filters, we'll get 6 separate activation maps:



Заметим, что третья размерность равна количеству фильтров.

У свертки есть два параметра: **stride** и **pad**.

Stride – шаг фильтра, т.е. если он 1, то фильтр размера h переберёт все подквадратики размера h. А если два, то он будет прыгать на 2. Обычно свертка происходит слева направо и снизу вверх.

Pad: параметр, определяющий на сколько можно "отступить" за границу.

Заметим, что если размер фильтра F, а картинка $N \cdot N$ и у нас есть некий stride S, а pad P, то Activation Map будет размера:

$$\frac{N - F + 2P}{S} + 1.$$

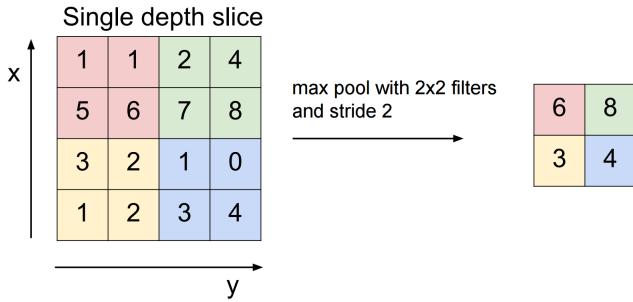
Важное следствие, если оставить делать свертки с $S = 1$, $P = 1$, то размерность ваших Activation Map будет снижаться. Мы этого не хотим. Тогда мы можем сделать свертки с $S = 1$, $P = 2$, проверяя формулу, мы увидим, что размерность тогда останется такой же, это хороший хак.

Насчет сверток 1x1 – иногда это круто, они используются, чтобы снизить размерность глубины. Напоминаю: размерность глубины = количеству фильтров.

3. Pooling

Иногда нам нужно понизить размерность тензора, сохраняя его главные особенности. Тут-то нам на помощь и приходит Pooling.

MAX POOLING



На картинке показан Max Pooling. Опишем более формально:

Пусть у нас есть тензор размера $H \cdot W \cdot D$. У Pooling есть параметры, stride (как у Conv) и размер.

После применения Pooling с параметрами S – stride размера F мы получим новый тензор с размерами $H_1 \cdot W_1 \cdot D$, где

$$H_1 = \frac{H - F}{S} + 1,$$

$$W_1 = \frac{W - F}{S} + 1.$$

Распространены значения $F = 2$, $S = 2$, чтобы понизить размерности высоты и ширины в 2 раза.

Обычно юзают либо Max Pooling, либо Average Pooling. (берут максимальное и среднее соответственно в квадратике $F \cdot F$).

4. Batch Normalization

Статейка по BN

В чём проблема?

Допустим, у вас многослойная глубокая сеть. Пусть вы учитёте её на каком-то минибатче X_{batch} . Рассмотрим слой 1 и слой 2. После backprop'а параметры слоя 1 могут очень сильно изменяться, таким образом вы сильно меняете распределение выхода даты на первом слое, а значит, второй слой должен к этому адаптироваться. Эта проблема называется **internal covariance shift**.

Даже для маленьких сетей скорость сходимости уменьшается, а для ультраглубоких сетей их сходимость под вопросом, т.к. эти проблемы накапливаются с количеством уровней. Поэтому, кстати, VGGNet по началу не училась :).

BatchNorm же призван пофиксить эту проблему.

Input: Values of x over a mini-batch: $\mathcal{B} = \{x_1 \dots m\}$;

Parameters to be learned: γ, β

Output: $\{y_i = BN_{\gamma, \beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m} \sum_{i=1}^m x_i \quad // \text{mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 \quad // \text{mini-batch variance}$$

$$\hat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \quad // \text{normalize}$$

$$y_i \leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i) \quad // \text{scale and shift}$$

Мы предполагаем, что размер батча адекватный (большой), поэтому между батчами распределения данных одинаковые, поэтому берётся батч и после каждого слоя FC или Conv нормализуется, считается среднее и выборочная дисперсия и преобразуется.

BN **сильно** ускоряет сходимость и является методом регуляризации (пояснять это я, конечно, не буду, вот [тут](#) подробнее).

11.3 Нелинейности (relu vs sigmoid, softmax)

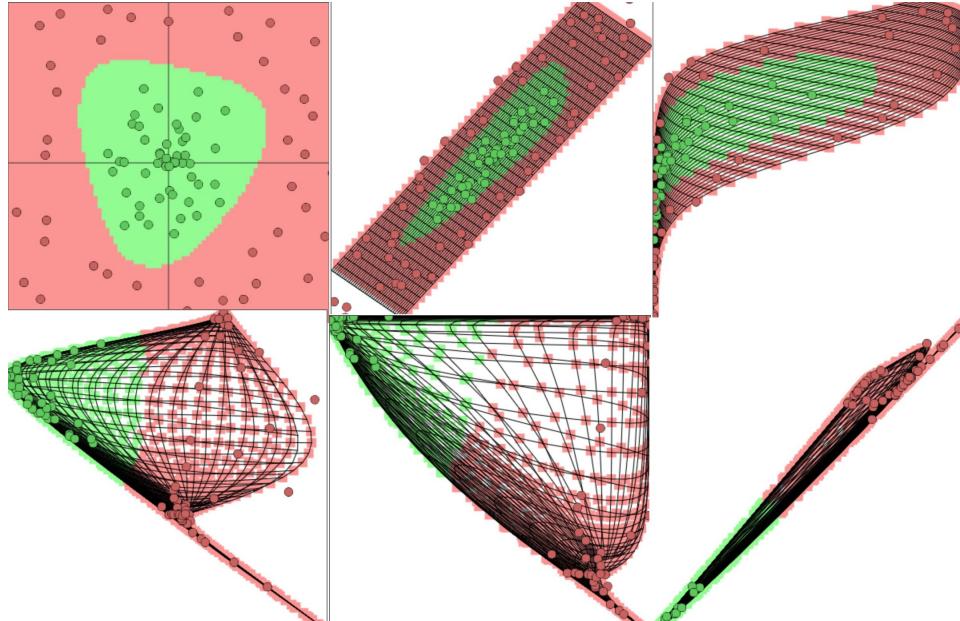
Когда мы умножаем на матрицу (FC layer), мы совершаём афинное преобразования пространства, но это не слишком круто, поэтому давайте введём нелинейности.

Фишкой нелинейностей в том, что они умеют искривлять пространство, делая наши данные линейно разделимыми.

[Вот сайтик.](#)

Поэтому без нелинейностей мы бы получили обычный линейный классификатор.

Рассмотрим сеть: InputLayer, FC(6), Активация tanh, FC(2), Активация tanh, SoftMax



Рассмотрим сеть уже после обучения.

Заметим, что мы смотрим на последние два нейрона, т.к. 6-мерное пространство представить себе тяжело.

На первой картинке мы видим наши данные. Дальше первый FC просто аффинно изменяет, а вот тангенс уже прикольно искривляет наше пространство на 3 картинке. Ещё 2 картинки пытаются как-то изменить пространство и в итоге в конце мы получаем линейноразделимую плоскость, которую наша сеть-классификатор легко делит на области :).

Перейдем к описанию нелинейностей:

Activation Functions

Sigmoid
 $\sigma(x) = \frac{1}{1+e^{-x}}$

Leaky ReLU
 $\max(0.1x, x)$

tanh
 $\tanh(x)$

Maxout
 $\max(w_1^T x + b_1, w_2^T x + b_2)$

ReLU
 $\max(0, x)$

ELU
 $\begin{cases} x & x \geq 0 \\ \alpha(e^x - 1) & x < 0 \end{cases}$

1. Sigmoid

Сжимает числа в $[0, 1]$, размазывая их по сигмойде. Однако есть проблемы: она нецентрирована, а главное она очень плохо обходится с большими отрицательными числами, поэтому gradient flow останавливается на ней, ведь они почти всегда около нуля.

2. tanh

Эта функция активации призвана решить проблему центрирования, но проблема градиента на ней остается, поэтому для глубоких сетей – очень плохой выбор.

3. ReLU

Эта функция активации очень хороша, потому что не сжимает числа (по крайней мере в положительной полуплоскости), производная **очень** быстро вычисляется, ведь это просто индикатор положительности. С такими функциями ваша сеть сходится примерно в 6 раз быстрее. Но она опять нецентрирована.

Ещё одна проблема – ваша сеть может обучится на неочень хороших данных и на всех хороших ReLU какого-то слоя всегда будет давать 0, т.е. умрёт, это плохо.

4. Leaky ReLU, ELU

Хипстерский ReLU, который решает последнюю проблему - ваши нейроны не умирают.

5. Softmax

Делает из вектора длины k вектор длины k . Аналог многомерной логистической функции. Выходы - типа вероятность принадлежности к классам. Обычно ставится в конце.

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

11.4 Функции потерь (logloss, l2, hinge)

LogLoss. Логарифмическая ошибка. Хорошо оценивает вероятность.

$$LogLoss = - \sum_{i=1}^l (y_i \ln p_i + (1 - y_i) \ln(1 - p_i)),$$

где y_i – реальная метка, p_i – предсказанная вероятность.

$$l2 = \frac{1}{n} \sum_{i=1}^n (\hat{Y}_i - Y_i)^2,$$

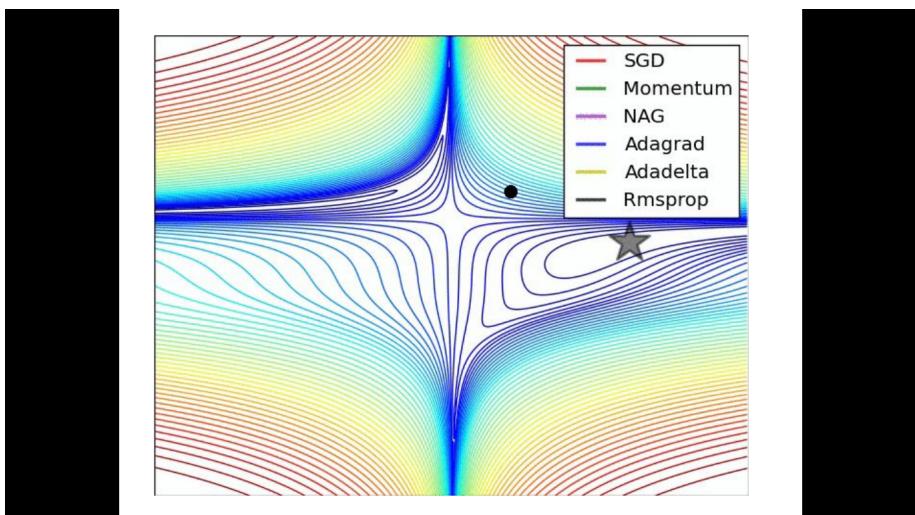
где \hat{Y} – предсказанный результат, Y реальный.

$$Hinge = \frac{1}{N} \sum_{i=1}^N \sum_{j \neq y_i} \max(0, s_j - s_{y_i} + 1),$$

N – количество объектов, y_i – правильный ответ для i -го объекта, s_j – ответ нашего алгоритма о принадлежности i -го объекта к j -ому классу.

12 Нейронные сети, обучение (backprop), оптимизация для нейронных сетей (sg, msg, nmsg, rmsprop, adam), регуляризация нейросетей (dropout, dropconnect, l1, l2, batchnorm)

12.1 Оптимизация:



1. SG:

SGD

$x_{t+1} = x_t - \alpha \nabla f(x_t)$ Плох тем, что можно застрять в локальном минимуме или седловой точке, так как там градиент нулевой. Также минус то, что зависит только от одного батча, появляется шум от батчей.

2. SGM:

SGD+Momentum

$$v_{t+1} = \rho v_t + \nabla f(x_t)$$

$$x_{t+1} = x_t - \alpha v_{t+1}$$

Продолжаем идти по направлению скорости v , которая каждый раз накапливается. ρ олицетворяет трение. Выбираемся из седловых точек и лок. минимумов.

3. NMSG (Nesterov Momentum)

Nesterov Momentum

$$v_{t+1} = \rho v_t - \alpha \nabla f(x_t + \rho v_t)$$

$$x_{t+1} = x_t + v_{t+1}$$

В SGM мы сначала делаем шаг в направлении скорости, а потом в направлении градиента, но казалось бы логично делать наоборот... мы же все равно шагнем, так почему бы еще раз шагнуть в сторону уменьшения в новой точке, а не старой.

4. RMSPROP

AdaGrad

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared += dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```



RMSProp

```
grad_squared = 0
while True:
    dx = compute_gradient(x)
    grad_squared = decay_rate * grad_squared + (1 - decay_rate) * dx * dx
    x -= learning_rate * dx / (np.sqrt(grad_squared) + 1e-7)
```

Для начала AdaGrad - его идея в том, чтобы "нормализовывать" шаги по всем направлениям. Чтобы не было огромных шагов по какому-то направлению. А RMSPROP, видимо, делает так, чтобы градиент совсем уж не затухал... То есть уменьшает значимость пройденного пути по направлениям.

5. Adam

Adam (full form)

```
first_moment = 0
second_moment = 0
for t in range(1, num_iterations):
    dx = compute_gradient(x)
    first_moment = beta1 * first_moment + (1 - beta1) * dx
    second_moment = beta2 * second_moment + (1 - beta2) * dx * dx
    first_unbias = first_moment / (1 - beta1 ** t)
    second_unbias = second_moment / (1 - beta2 ** t)
    x -= learning_rate * first_unbias / (np.sqrt(second_unbias) + 1e-7)
```

Momentum

Bias correction

AdaGrad / RMSProp

Bias correction for the fact that
first and second moment
estimates start at zero

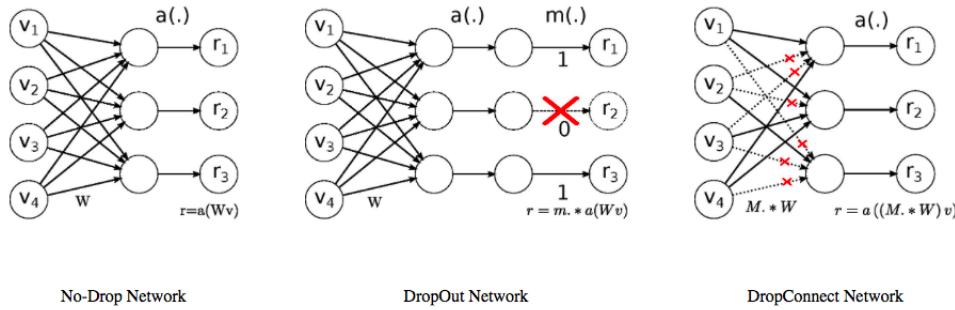
Adam with $\beta_1 = 0.9$,
 $\beta_2 = 0.999$, and $\text{learning_rate} = 1e-3$ or $5e-4$
is a great starting point for many models!

По сути adam - это RMSPROP + Momentum. Bias correction позволяет справится с тем, что в начале накопленное среднее еще не очень большое (на первой итерации вообще 0). Несложно заметить, что с этой поправкой, например, на первой итерации $first_unbias$ просто равен dx . Эта поправка необходима лишь вначале, и, собственно, при $t \rightarrow \infty$ она почти никак и не влияет.

12.2 Регуляризация

Может случится так, что сеть переобучится, то есть найдет зависимости в тренировочной выборке, которых нет в генеральной совокупности или попадет в локальный минимум, далекий от оптимального. Для борьбы с этим есть несколько стандартных решений в виде добавления доп. слоев для регуляризации (Noise Layers, Regularisation Layers).

- **Dropout** - обнуляет каждый элемент предыдущего слоя с некоторой фиксированной вероятностью, тем самым уменьшая количество связей
- **Dropconnect** - аналогично dropout, но действует не на выходы нейронов, а на ребра, то есть случайно обнуляет каждый переход (элемент матрицы переходов), обычно применяется в полно связанных слоях.



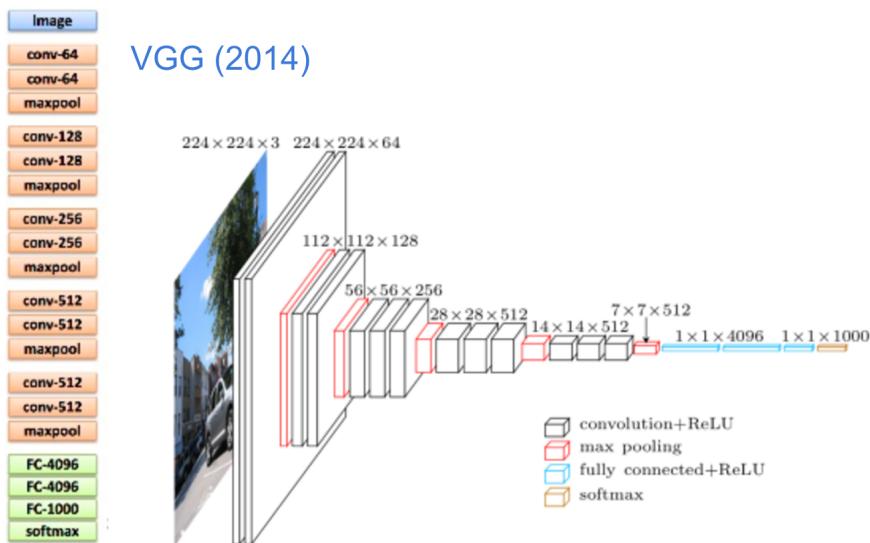
• **l1, l2** - функции сопоставляющие одному или нескольким слоям штраф за большие значения весов (вычисляются по соответствующим формулам: $\lambda \sum_i |w_i|$, $\lambda \sum_i |w_i|^2$), для минимизации добавляются к общему лоссу сети (большие коэффициенты сети также можно считать переобучением)

• **batchnorm** - слой нормирующий входы: вычитает среднее по батчу и делит на выборочную дисперсию (покоординатно)

$$y_i = \frac{x_i - \mu}{\sqrt{\sigma^2 + \epsilon}} + \beta$$

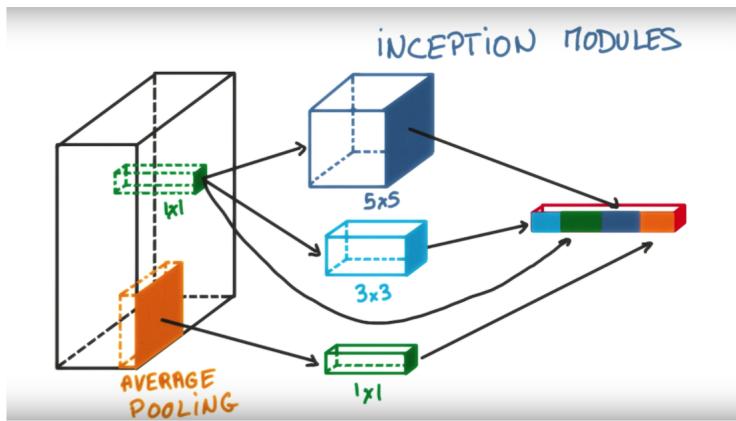
13 Нейронные сети, обучение (backprop), современные сверточные нейронные сети (vgg, resnet, inception) и детали обучения (batchnorm, pretraining)

13.1 VGG



VGG - свёрточная сеть из 2014 года, основанная на куче свёрток 3x3, а затем пуллингов. Её изобрели ещё до появления BatchNorm, поэтому с обучением были сложности. Самой глубокой сетью была VGG-19, но обучить её сходу не получалось, поэтому сначала обучали сеть гораздо меньшего размера, затем VGG-16, затем её. Таким образом, они смогли перебороть плохую инициализацию и сеть начала учиться. Далее, они изобрели BatchNorm и сеть смогла учиться и без предобучения маленькими сетями.

13.2 Inception



GoogleNet (2014)



"Large but fast":

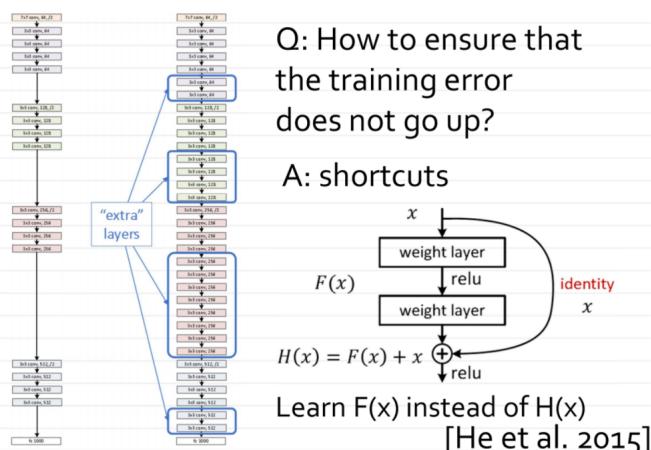
- Keep large number of maps
- apply convolutions only to dimensionality reduced stacks



Inception - свёрточная сеть от гугла, вышла следом за VGG. Люди подумали, почему мы делаем только 3×3 свёртки? И сделали все свёртки, соединив их (взяв конкатенацию) в FC или Pool Layer. Появилась проблема - сети стали слишком глубокими и градиент затухал, тогда Google сделали эту сеть трёхголовой, что решает эту проблему. Ведь теперь самые первые уровни сети изменяются не только под действием одного (самого глубокого) выхода, а всех трёх на разной глубине.

13.3 ResNet

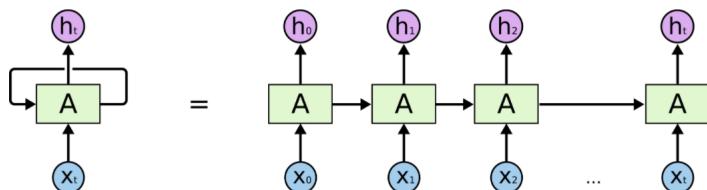
ResNet (2015)



ResNet - одна из самых современных свёрточных сетей от Microsoft. Чем глубже мы делаем сети, тем острее встаёт проблема затухания градиента. В резнете это решено с помощью шорткатов - специальных связей, которые позволяют градиенту (проверьте сами backprop выкладку), по x (на картинке) проходить дальше, независимо от градиента $F(x)$, тем самым если в нашем $F(x)$ блоке затухнет градиент, то в итоге сеть не перестанет учиться.

14 Рекуррентные НС, обучение (backprop tt), отличие от сверточных, разновидности рекуррентных слоев (RNN, LSTM, GRU) аннотация изображений, перевод, диалоговые системы

14.1 Рекуррентные НС



An unrolled recurrent neural network.

Рекуррентные нейронные сети состоят из последовательности скрытых состояний, в которые ведут входы x_i и выходы h_i . Таким образом они могут принимать на вход последовательности из нескольких объектов. Например, тексты. Формула для очередного состояния:

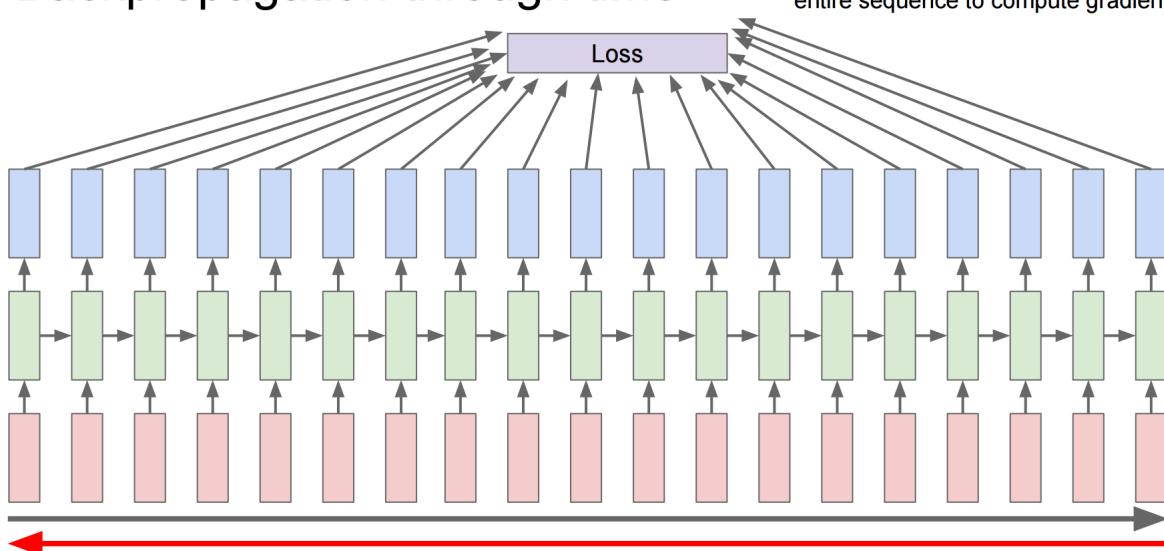
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t), \quad y_t = W_{hy}h_t.$$

Таким образом, мы умножаем аутпут предыдущего состояния на матрицу, а затем прибавляем к ней текущий вектор умноженный на другую матрицу весов, накладываю на всё это нелинейность.

14.2 Backprop TT

Backpropagation through time

Forward through entire sequence to compute loss, then backward through entire sequence to compute gradient



Backpropagation through time

Сначала мы делаем forward pass по всей последовательности, а затем backward pass по опять всей последовательности. Мы "развертываем" рекуррентную сеть в обычную feed-forward, но чтобы посчитать производную на шаге backpropa, мы должны сложить все производные и обновить матрицу W_{hh} , тут кроется проблема. Мы очень много раз применим коррелированные обновления, а это очень плохо оказывается на SGD.

Появляется программа взрывающихся или затухающих градиентов.

14.3 Vanishing Gradient

Рассмотрим без активационной функции для упрощения математики, хотя с ней ничего не изменяется.

$$h_t = Wf(h_{t-1}) + W^{hx}x_t, \quad y_t = W^{hy}f(h_t).$$

Тогда как было выше сказано, ошибка это сумма ошибок

$$\frac{\partial E_t}{\partial W} = \sum_{i=1}^T \frac{\partial E_t}{\partial W}.$$

Chain rule:

$$\frac{\partial E_t}{\partial W} = \sum_{k=1}^t \frac{\partial E_t}{\partial y_t} \frac{\partial y_t}{\partial h_t} \frac{\partial h_t}{\partial h_k} \frac{\partial h_k}{\partial W}.$$

Посмотрим на $\frac{\partial h_t}{\partial h_k}$, разложим её дальше.

$$\frac{\partial h_t}{\partial h_k} = \prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}}.$$

Каждая такая производная - это матрица (Якобиан).

Проанализируем норму якобиана.

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| \leq \|W^T\| \cdot \|diag[f'(h_{j-1})]\| \leq \beta_W \beta_h,$$

где β - верхняя граница нормы.

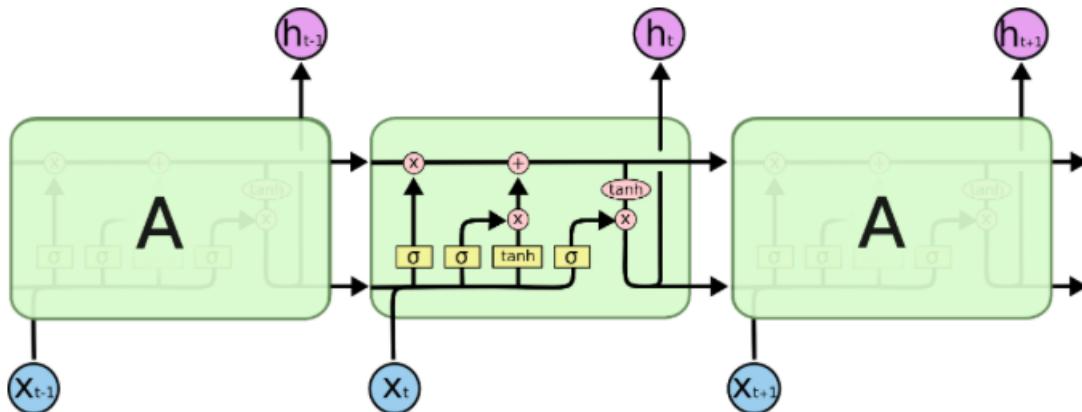
Тогда

$$\left\| \frac{\partial h_t}{\partial h_k} \right\| = \left\| \prod_{j=k+1}^t \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq (\beta_w \beta_h)^{t-k}.$$

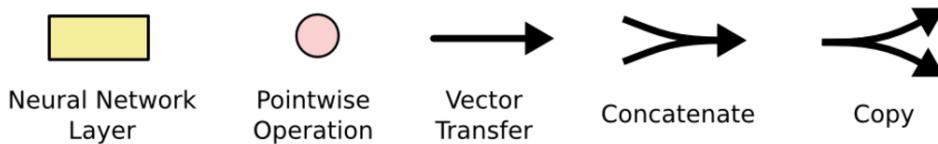
Функция экспоненциальна от длины последовательности, да ещё и экспонента, поэтому в Backprop TT очень часто затухает, или наоборот - взрывается, градиент.

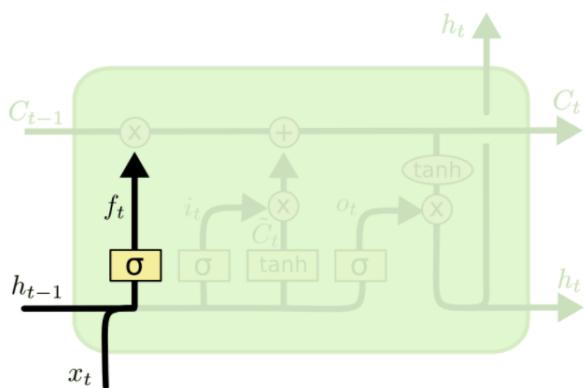
Обычно в vanilla RNN ограничивают длину последовательности, по которой делают BPRTT, и делают gradient clipping - ограничивают градиент, если он выше какого-то значения. Однако от затухающих градиентов это не спасает, однако спасает LSTM.

14.4 LSTM



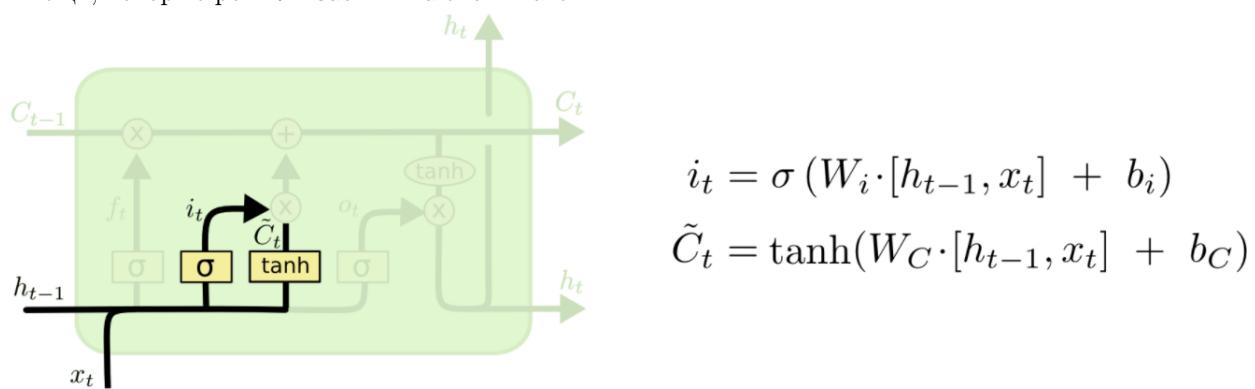
Ключевая идея - будем хранить в каждом состоянии сети новую величину c_t , которую будем также прокидывать в дальнейшие состояния. Верхняя линия передаёт c_t через все блоки, поэтому она сохраняет "память".





$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

Сначала мы берём h_t с предыдущего слоя и текущий инпут, и производим преобразование f_t . Это преобразование называется forget layer, оно решает, добавим ли мы значение в c_t или нет. Сигмойда для каждого значения выдает значения от 0 до 1, где 0 - полностью "забыть" значение, а 1 "полностью" добавить. Когда мы умножим C_{t-1} на f_t мы "забудем" вещи, которые решили забыть на этом шаге.

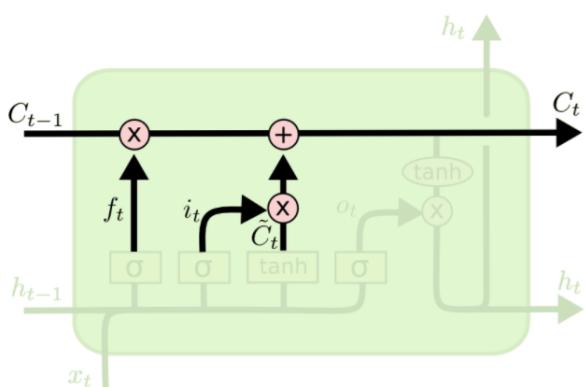


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

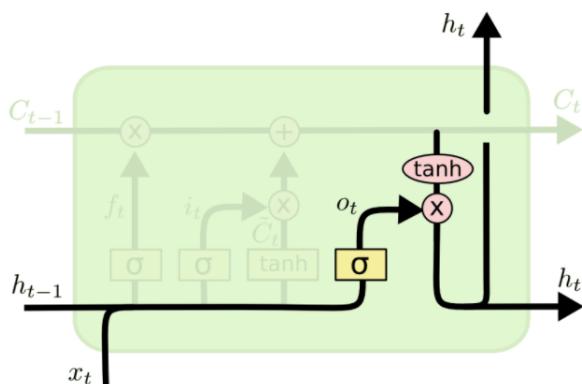
Далее есть i_t - input layer, и \tilde{C}_t - candidate layer. Input Layer решает, какие значения мы обновим, а какие оставим. А Candidate layer решает, какие значения нам нужно будет добавить в c_t - cell state.

Теперь мы добавляем $i_t \cdot \tilde{C}_t$ (новые значения, уже умноженные на i_t - коэффициент, насколько мы хотим обновить значения).



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

На этом шаге мы изменяем c_t , добавляя новое и одновременно забывая предыдущее.



$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t * \tanh(C_t)$$

Теперь мы прокидываем наш c_t дальше, а потом сжимаем c_t с помощью \tanh в $(-1; 1)$, умножая при этом на o_t , где o_t значит, насколько мы хотим "забыть" наши значения \tilde{C}_t .

В итоге у этой сети получается очень много параметров, однако она невероятно мощна, потому что сеть сама понимает, когда нужно забыть что-то, а когда наоборот, оставить все.

GRU: (вроде, в лекциях не было) Там два гейта, а не три как в LSTM.

• denotes the Hadamard product. $h_0 = 0$.

$$z_t = \sigma_g(W_z x_t + U_z h_{t-1} + b_z)$$

$$r_t = \sigma_g(W_r x_t + U_r h_{t-1} + b_r)$$

$$h_t = z_t \circ h_{t-1} + (1 - z_t) \circ \sigma_h(W_h x_t + U_h (r_t \circ h_{t-1}) + b_h)$$

Variables

- x_t : input vector
- h_t : output vector
- z_t : update gate vector
- r_t : reset gate vector
- W , U and b : parameter matrices and vector

Activation functions

- σ_g : The original is a sigmoid function.
- σ_h : The original is a hyperbolic tangent.

14.5 Аннотация изображений, перевод, диалоговые системы

Есть энкодер (в случае перевода RNN), принимающий на вход некую последовательность (в переводе — текст на первом языке). Скрытое состояние с последней стадии можно интерпретировать как признаки этой последовательности. Эти признаки запихиваем в декодер (тоже RNN) и получаем выходную последовательность (в переводе — текст на втором языке).

$$seq_1 \rightarrow encoder \rightarrow decoder \rightarrow seq_2$$

= (

15 Задача снижения размерности пространства признаков. Идея метода главных компонент (PCA). Связь PCA и сингулярного разложения матрицы признаков (SVD).

Идея в максимально простом виде: давайте сменим базис, чтобы разброс объектов по первым координатам был больше чем по последним. Тогда скорее всего для наших задач хватит нескольких первых координат.

Более подробно:

Сингулярным разложением матрицы M порядка $m \times n$ является разложение следующего вида

$X = U\Sigma V^T$, где Σ — матрица размера $m \times n$ с неотрицательными элементами, у которой элементы, лежащие на главной диагонали — это сингулярные(собственные) числа (а все элементы, не лежащие на главной диагонали, являются нулевыми), а матрицы U (порядка m) и V (порядка n) — это две унитарные матрицы, состоящие из левых и правых сингулярных векторов соответственно (а V^T — это сопряжённо-транспонированная матрица к V).

Кроме того в Σ сингулярные числа идут по убыванию ($\lambda_1 > \dots > \lambda_{\min(m,n)}$). И чем больше сингулярное число, тем больший разброс объектов вдоль компоненты - соответствующего сингулярному числу столбца в матрице U .

Чтобы отобрать k главных компонент, мы берем первые (наибольшие) k сингулярных чисел, потом соответствующие им столбцы матрицы U . И всё, эти столбцы $c_1 \dots c_k$ и есть главные компоненты. (Кстати, столбцы матрицы U - собственные векторы матрицы $X^T X$.

У пространства натянутого на эти компоненты есть несколько хороших свойств.

- 5) Пусть M_k – это подпространство, натянутое на главные оси c_1, \dots, c_k . Оказывается, при проецировании объектов на произвольное подпространство L_k размерности k в \mathbb{R}^m геометрическая структура *искажается в наименьшей степени*, если этим подпространством является M_k (см. [1, с. 350]):
- сумма квадратов расстояний от объектов до их проекций на L_k минимальна, когда $L_k = M_k$ (в этом случае она равна $n(\lambda_{k+1} + \dots + \lambda_m)$ (доказательство см. в [76, с. 243]));
 - при проецировании на M_k наименее искажается сумма квадратов расстояний между всевозможными парами объектов (для M_k ее изменение составляет $n^2(\lambda_{k+1} + \dots + \lambda_m)$);
 - когда $L_k = M_k$, в наименьшей степени искажаются расстояния от объектов до их центра масс (совпадающего с началом координат **0** ввиду допущения D1), а также углы между всевозможными парами прямых, соединяющих объекты с **0**.

(Лагутин, Наглядная математическая статистика, с 321. Там довольно понятно про PCA)

16 Вычисление SVD в пространствах высокой размерности методом стохастического градиента (SG SVD).

Тут что-то есть. Пытаюсь разобраться

SVD: $A = USV^T$

Нам нужно как-то считать его в пространствах больших размерностей.

$$A = USV^T = (US^{\frac{1}{2}})(S^{\frac{1}{2}}V^T) = (X)(Y^T) = XY^T$$

Давайте запилим задачу оптимизации:

$$L(A, X, Y, W, \lambda) = \sum_{i=1}^m \sum_{j=1}^n w_{ij} \rho(A_{ij}, x_i y_i^T) + \lambda \sum_{i=1}^m x_i x_i^T + \lambda \sum_{j=1}^n y_j y_j^T$$

Здесь W - веса для разных элементов матрицы A (хз, зачем они нужны). λ - коэффициент регуляризации.

На эту задачу уже можно натравить SGD (stochastic gradient descent).

Матрица S из X и Y достаточно легко вытаскивается. $X = US^{\frac{1}{2}}$, столбцы U нормированы. Значит корни из λ_i это нормы столбцов X . То есть U, S, V^T восстанавливаются без проблем. Проверьте кто-нибудь!

17 Идея методов SNE, tSNE, принципиальные отличия от PCA.

Идеи алгоритмов:

PCA - давайте сменим базис, чтобы разброс объектов по первым координатам был больше чем по последним. Тогда скорее всего для наших задач хватит нескольких первых координат.

SNE - давайте натянем "пружинки" между объектами, которые хотели бы, чтобы расстояния между объектами были как в исходном пространстве. Потом запихнем эти объекты в пространство меньшей размерности и посмотрим под действием "пружинок" расположатся объекты.

Про SNE и tSNE

Про PCA (с 89)

Принципиальные отличия:

1. PCA линеен (в связи с чем может меньше), SNE и tSNE нет
2. SNE и tSNE неинтерпретируемые, а в PCA первые несколько главных компонент как правило можно интерпретировать
3. Добавьте, чего тут нет

18 Задача кластеризации. Агglomerативная и дивизионная кластеризация.

Про кластеризацию (с 113)

Задача кластеризации (unsupervised or semi-supervised) заключается в следующем. Имеется обучающая выборка $X^l = \{x_1, \dots, x_l\} \subset X$ и функция расстояния между объектами $\rho(x, x')$. Требуется разбить выборку на непересекающиеся подмножества, называемые кластерами, так, чтобы каждый кластер состоял из объектов, близких по метрике ρ , а объекты разных кластеров существенно отличались. При этом каждому объекту $x_i \in X_l$ приписывается метка (номер) кластера y_i

Про агglomerативную и дивизионную кластеризацию (с 122).

Иерархические алгоритмы кластеризации, называемые также алгоритмами таксономии, строят не одно разбиение выборки на непересекающиеся классы, а систему вложенных разбиений. Результат таксономии обычно представляется в виде таксономического дерева — дендрограммы.

Классическим примером такого дерева является иерархическая классификация животных и растений. Среди алгоритмов иерархической кластеризации различаются два основных типа.

1. Дивизионные или нисходящие алгоритмы разбивают выборку на всё более и более мелкие кластеры.
2. Агglomerативные или восходящие алгоритмы, в которых объекты объединяются во всё более и более крупные кластеры.

19 Кластеризация с помощью ЕМ-алгоритма (без вывода М-шага). Формула Ланса-Уилльямса.

Кластеризация с помощью ЕМ-алгоритма (с 119)

Алгоритм 7.3. Кластеризация с помощью ЕМ-алгоритма

1: начальное приближение для всех кластеров $y \in Y$:

$$w_y := 1/|Y|;$$

μ_y := случайный объект выборки;

$$\sigma_{yj}^2 := \frac{1}{\ell|Y|} \sum_{i=1}^{\ell} (f_j(x_i) - \mu_{yj})^2, \quad j = 1, \dots, n;$$

2: **повторять**

3: Е-шаг (expectation):

$$g_{iy} := \frac{w_y p_y(x_i)}{\sum_{z \in Y} w_z p_z(x_i)}, \quad y \in Y, \quad i = 1, \dots, \ell;$$

4: М-шаг (maximization):

$$w_y := \frac{1}{\ell} \sum_{i=1}^{\ell} g_{iy}, \quad y \in Y;$$

$$\mu_{yj} := \frac{1}{\ell w_y} \sum_{i=1}^{\ell} g_{iy} f_j(x_i), \quad y \in Y, \quad j = 1, \dots, n;$$

$$\sigma_{yj}^2 := \frac{1}{\ell w_y} \sum_{i=1}^{\ell} g_{iy} (f_j(x_i) - \mu_{yj})^2, \quad y \in Y, \quad j = 1, \dots, n;$$

5: Отнести объекты к кластерам по байесовскому решающему правилу:

$$y_i := \arg \max_{y \in Y} g_{iy}, \quad i = 1, \dots, \ell;$$

6: **пока** y_i не перестанут изменяться;

Хорошо работает кластеров с распределениями похожими на распределения эллиптических гауссианов. Соответственно не очень хорошо со всем остальным

Для понимания идеи ЕМ:

Как ЕМ помогает в кластеризации? ЕМ-алгоритм начинает с того, что пытается сделать вывод на основании параметров модели.

Затем следует итерационный трехшаговый процесс:

- Е-шаг: На этом шаге на основании параметров модели вычисляются вероятность принадлежности каждой точки данных к кластеру.
 - М-шаг: Обновляет параметры модели в соответствии с кластерным распределением, проведенным на шаге Е.
 - Предыдущие два шага повторяются до тех пор, пока параметры модели и кластерное распределение не уравняются.
- Формула Ланса-Уильямса** (с 122)
- Формула Ланса-Уильямса нужна для быстрого пересчета межкластерных расстояний при слиянии кластеров в агломеративных алгоритмах.

$$R(U \cup V, S) = \alpha_U R(U, S) + \alpha_V R(V, S) + \beta R(U, V) + \gamma |R(U, S) - R(V, S)|,$$

Коэффициенты зависят от используемой метрики. Эти коэффициенты существуют для почти всех разумных метрик.

20 Алгоритмы k-Means и DBSCAN.

Демо алгоритмов: [k-Means demo](#), [DBSCAN demo](#)

Теория:

[k-Means](#) (с 120)

Алгоритм 7.4. Кластеризация с помощью алгоритма k -средних

- 1: сформировать начальное приближение центров всех кластеров $y \in Y$:
 μ_y — наиболее удалённые друг от друга объекты выборки;
- 2: **повторять**
- 3: отнести каждый объект к ближайшему центру (аналог Е-шага):
 $y_i := \arg \min_{y \in Y} \rho(x_i, \mu_y), \quad i = 1, \dots, \ell;$
- 4: вычислить новое положение центров (аналог М-шага):
 $\mu_{yj} := \frac{\sum_{i=1}^{\ell} [y_i = y] f_j(x_i)}{\sum_{i=1}^{\ell} [y_i = y]}, \quad y \in Y, \quad j = 1, \dots, n;$
- 5: **пока** y_i не перестанут изменяться;

Плюсы:

1. Интуитивно понятный, дает хорошие результаты в очевидных случаях.
1. Не гарантируется достижение глобального минимума суммарного квадратичного отклонения V , а только одного из локальных минимумов.
2. Результат зависит от выбора исходных центров кластеров, их оптимальный выбор неизвестен.
3. Число кластеров надо знать заранее.
4. Рассчитан на шары. Плохо работает для кластеров сложной формы....

DBSCAN

Алгоритм:

- Назовем точку p как core point, если у нее есть как минимум minPts соседей, включая ее саму, на расстоянии не больше ε . Назовем этих соседей прямо-достижимыми из точки p . По определению все точки могут быть прямо-достижими только из core points.

- Точка q достижима из точки p , если существует цепочка p_1, \dots, p_n с $p_1 = p$ и $p_n = q$, где каждый p_{i+1} прямо-достижим из p_i (все точки цепочки должны быть core points, кроме может быть q).

- Все точки недостижимые из других точек назовем выбросами.

- Если p - core point, то она образует кластер со всеми достижимыми из нее точками.

Плюсы:

1. Не нужно указывать количество кластеров
2. Форма не имеет значения. (Справляется с ленточными кластерами.)
3. Устойчив к шумам и выбросам

4. У алгоритма всего два параметра. minPts и ε
5. Дает почти детерминированный результат с точностью до перенумерования кластеров и определения принадлежности выбросов и граничных точек. (Не меняется от запуска к запуску)
6. Работа алгоритма может быть ускорена при использовании структур данных, поддерживающих объемные (в шаре) запросы.
7. Параметры алгоритма может предсказать эксперт.

Минусы:

1. Всё-таки не совсем детерминированный. Неоднозначность определения принадлежности выбросов и граничных точек.
2. Сложно подобрать значение ε и метрику. Особенно в пространствах большой размерности.
3. Не умеет в кластеры разной плотности. Так как minPts и ε одинаковые для всех кластеров.

Часть II

Доп. вопросы НЕ ПИШИТЕ ИХ

21 Основные понятия

1. Что такое задачи классификации, кластеризации и регрессии? Какие из них относятся к supervised learning, а какие - к unsupervised?
2. Что такое переобучение и недообучение? Как их можно детектировать?
3. Что такое обучающая и тестовая выборки, кросс-валидация? Как устроена k-fold cross validation?

22 Простые методы

1. Как работает kNN в задаче классификации?
2. Как работает kNN с весами объектов в задаче классификации и в задаче регрессии?
3. Как работает наивный байесовский классификатор, в чем заключается его наивность?
4. Как приближается исходная зависимость y от x в линейной регрессии и как настраиваются веса в ней?

23 Метрики качества в задачах классификации и регрессии

1. Как вычисляются и в каких задачах (классификации/регрессии) применяются метрики: accuracy, precision, recall, F1-measure, ROC-AUC, log loss, MSE, MAE, RMSE?
2. Решается задача бинарной классификации (с двумя классами 0 и 1), в которой пр-меры из класса 0 составляют 95% выборки. Какие метрики из перечисленных в преды-дущем вопросе предпочтительней использовать?
3. К оценке какой величины для распределения y при условии x приводят MSE и MAE?
4. Можно ли при таргетах из множества $Y = \{0; 1\}$ использовать для оценки $P(y=1|x)$ не log loss, а MSE?

24 Деревья

1. Как выглядит решающее дерево? Как применяется уже построенное для задачи классификации дерево? А для задачи регрессии?
2. Как строятся решающие деревья? (рекомендуется обратиться к материалам лекций или документации sklearn)
3. Как выглядят энтропийный критерий, критерий Джини и среднеквадратичное отклонение, используемое как критерий в задаче регрессии?

4. Что такое node impurity и goodness of split? Как они связаны?
5. Какие преимущества и недостатки есть у деревьев? (полезно как подумать самостоятельно, так и обратиться к документации sklearn)
6. Есть ли разница (с точки зрения вида получаемого в итоге дерева): строить каждое разбиение в дереве, максимизируя информативность, или строить каждое разбиение, минимизируя "ошибку как было предложено на первой лекции про деревья"?

25 Общие идеи построения композиций

1. Что такое bagging, blending, stacking, boosting?
2. Нужно ли как-то делить выборку, чтобы избежать переобучения, при реализации стэкинга?
3. В чем преимущества и недостатки бустинга и бэггинга?

26 Градиентный бустинг

1. В чем основная идея градиентного бустинга?
2. Как выглядит алгоритм градиентного бустинга в самом общем виде — с произвольной функцией потерь в функционале ошибки и произвольным функционалом, оценивающим качество приближения антаградиента?
3. Как выглядит алгоритм градиентного бустинга с квадратичными функциями потерь? На что настраиваются базовые алгоритмы?
4. Как выглядит алгоритм градиентного бустинга в случае задачи бинарной классификации?
5. Какие параметры есть у классификаторов и регрессоров на основе градиентного бустинга над деревьями в sklearn и XGBoost? Какие параметры стоит настраивать в первую очередь?
6. Какая высота деревьев оправдана в градиентном бустинге над деревьями? Почему?
7. Есть ли у градиентного бустинга склонность к сильному переобучению при увеличении количества деревьев? С какими еще параметрами алгоритма эффект переобучения может быть связан?
8. Какие есть методы борьбы с переобучением, применяемые в градиентном бустинге?

27 Случайный лес

1. Как работает Random Forest?
2. Зачем в Random Forest делается рандомизация с выбором подмножества признаков в каждом сплите?
3. Какой высоты деревья стоит строить в Random Forest?
4. Какие параметры есть у Random Forest в sklearn? Какие параметры стоит настраивать в первую очередь?
5. Есть ли у Random Forest склонность к сильному переобучению при увеличении количества деревьев? С какими еще параметрами алгоритма эффект переобучения может быть связан?
6. Как работает ExtraTreesClassifier из sklearn?

28 Linear models

1. Как выглядит решающее правило в линейной классификации? А зависимость, которой мы приближаем ответы в линейной регрессии?
2. Что такое функции потерь в задачах классификации и регрессии? Зачем они нужны?
3. Что такое регуляризаторы? Какими они бывают в задачах линейной классификации и регрессии? Зачем они нужны?
4. Как в общем виде выглядит оптимизационная задача в линейной классификации или линейной регрессии?

5. Как работает настройка весов в линейной модели с помощью SGD (Stochastic Gradient Decent)? Как выглядит правило обновления весов?
6. Учитывается ли коэффициент сдвига w_0 в регуляризаторе? Почему?
7. Почему линейные модели рекомендуется применять к выборке с нормированными значениями признаков?
8. Как выглядит оптимизационная задача в логистической регрессии? А в
9. Выпишите и докажите формулу для весов в линейной регрессии (с квадратичной функцией потерь). То же самое для гребневой регрессии.
10. Выпишите SGD для логистической регрессии с ℓ_2 -регуляризацией и для SVM с линейным ядром.
11. В чем заключается идея ядер в SVM?
12. Какие преимущества и недостатки есть у линейных моделей?

29 Neural networks

1. Чем нейросети отличаются от линейных моделей а чем похожи?
2. В чем недостатки полносвязанных нейронных сетей какая мотивация к использованию свёрточных?
3. Какие слои используются в современных нейронных сетях? Опишите как работает каждый слой и свою интуицию зачем он нужен.
4. Может ли нейросеть решать задачу регрессии, какой компонент для этого нужно заменить в нейросети из лекции 1?
5. Почему обычные методы оптимизации плохо работают с нейросетями? А какие работают хорошо? Почему они работают хорошо?
6. Для чего нужен backprop, чем это лучше/хуже чем считать градиенты без него? Почему backprop эффективно считается на GPU?
7. Почему для нейросетей не используют кросс валидацию, что вместо неё? Можно ли ее использовать?
8. Чем отличаются современные сверточные сети от сетей 5 летней давности?
9. Какие неприятности могут возникнуть во время обучения современных нейросетей?
10. У вас есть очень маленький датасет из 100 картинок, классификация, но вы очень хотите использовать нейросеть, какие неприятности вас ждут и как их решить? что делать если первый вариант решения не заработает?
11. Можно ли использовать сверточные сети для классификации текстов? Если нет обоснуйте :D, если да то как? как решить проблему с произвольной длинной входа?
12. Чем LSTM лучше/хуже чем обычная RNN?
13. Выпишите производную $\frac{dc_{n+1}}{dc_k}$ для LSTM ([подробнее](#)), объясните формулу, когда производная затухает, когда взрывается?
14. Зачем нужен ТВРТТ почему ВРТТ плох?
15. Как комбинировать рекуррентные и сверточные сети, а главное зачем? Приведите несколько примеров реальных задач.
16. Объясните интуицию выбора размера эмбединг слоя? почему это опасное место?

30 Unsupervised learning

1. В чём заключается проблема мультиколлинеарности?
2. Какие проблемы при обучении алгоритмов возникают из-за большой разамерности пространства признаков?
3. В чём суть проклятия размерности?
4. Какая связь между решением задачи PCA и SVD-разложение матрицы регрессии?
5. Почему в tSNE расстояние между парамми объектов измеряется "по-стъюденту" и как это помогает решить проблему "скрученности"(crowding problem)?
6. На какой идее базируются алгоритмы агломеративной кластеризации? Напишите формулу Ланса-Вильма
7. Какие два шага выделяют в алгоритме кластеризации k-means?
8. В чём отличия (основные упрощения) k-means от EM-алгоритма кластеризации?
9. Какой принцип работы графовых алгоритмов кластеризации?
10. В чём некорректность постановки задачи кластеризации?
11. Какие 3 интерпретации оптимизационной задаче можно предложить в методе PCA?

31 Теоретический минимум

1. В чём разница между задачами классификации, кластеризации, регрессии, уменьшения размерности, приведите примеры.
Не очень представляю, что имеется в виду, ведь задачи очень сильно отличаются друг от друга и сравнивать их тяжело. Поэтому напишу вкратце что это за задачи и пример.
 - (a) Задача классификации - задача обучения с учителем. У нас есть набор классов, для некоторого множества объектов есть ответы (знаем к какому классу они принадлежат), для некоторого другого множества нужно предсказать класс. Пример: предсказание вернет клиент банка кредит или нет по историческим данным.
 - (b) Задача кластеризации - задача обучения без учителя. Есть множество объектов нужно разбить их на группы так, чтобы "похожие" объекты оказались в одной, а непохожие в разных. Пример: есть разнородное множество объектов, для которых нужно решать какую-нибудь задачу, и хочется разбить его на кластера, чтобы в дальнейшем работать с ними по отдельности. Если еще конкретней, то можно рассмотреть рекомендацию товаров в магазине одежды. Ясно, что парням и девушкам нужно показывать разные рекомендации, поэтому множество потенциальных покупателей было бы неплохо разбить на кластеры.
 - (c) Регрессия - задача обучения с учителем, в которой есть выборка объектов, с известным значением вещественной целевой функции и выборка объектов, для которых это целевое значение нужно предсказать. Предполагаем, что целевая функция - функция признаков объекта и некоторого небольшого шума (желательно белого). В нашем курсе рассматривалась в основном линейная регрессия - регрессия, в которой предполагается, что эта зависимость линейная. Пример использования: классификация текстов - признаки набор пграмм.
 - (d) Уменьшение размерности - задача обучения без учителя, в которой хочется построить отображение из многомерного пространства в пространство существенно меньшей размерности с минимальными "потерями". Хотим либо уменьшить хорошо восстанавливать объекты обратно в многомерное пространство, либо чтобы в новом пространстве "похожие" объекты оказались близко, а "непохожие" далеко. Пример: есть большое количество признаков, многие из которых избыточны (например, они могут быть линейнозависимы), и мы хотим избавиться от лишних признаков.
2. Что такое объект, целевая переменная, признак, модель, функционал ошибки и обучение?

В наиболее общем виде задача машинного обучения (с учителем) заключается в восстановлении отображения из множества объектов в множество значений целевой переменной. Соответственно, объект - это то, что дано на входе. Используется признаковое описание объекта, когда ему сопоставляется набор различных характеристик: числовых, логических, категориальных, символьных и прочих. Предполагается, что эти признаки позволяют определить все существенные свойства объекта. Соответственно, можно отождествить объект с его признаковым описанием, тогда его можно представить как кортеж из значений признаков или элемент декартова произведения множеств значений для каждого признака. Например, если все признаки числовые, это просто вектор. Целевая переменная - образ объекта при искомом отображении. Как правило известно её множество значений: для задачи классификации это конечное множество (номера или названия классов), для задачи регрессии \mathbb{R} , \mathbb{R}^n или их подмножество. Модель

можно представлять как множество отображений, среди которых ведётся поиск, и способ этого поиска. Функционал ошибки - это числовая функция из декартова квадрата множества значений целевой переменной. Практически всегда берётся неотрицательная функция, равная нулю при равенстве её аргументов. Можно также потребовать выполнение свойств метрики. Функционал ошибки служит для оценки соответствия модели искуому отображению через сравнение предсказанного и истинного значения целевой переменной. Обучение - поиск отображения, сопоставляющего объекту значение целевой переменной. В задаче обучения с учителем происходит подгон параметров модели под выборку из пар (объект, целевая переменная).

3. Запишите формулы для линейной модели регрессии и для среднеквадратичной ошибки.

x_i - вектор признаков i -го объекта, y_i - значения целевой функции для i -го объекта.

$$f(x) = \langle x, w \rangle + w_0$$

$$RSS = \frac{1}{n} \sum_{i=1}^n n(f(x_i) - y_i)^2$$

4. Что такое градиент? Какое его свойство используется при минимизации функций?

Пусть $f : \mathbb{R}^n \rightarrow \mathbb{R}$.

$$\text{grad } f(x) = \left(\frac{\partial f}{\partial x_1}, \dots, \frac{\partial f}{\partial x_n} \right)$$

Свойство для минимизации: $-\text{grad } f(x)$ — направление наибольшего убывания функции.

5. Запишите формулу для одного шага градиентного спуска. Как модифицировать градиентный спуск для очень большой выборки? Почему в задаче классификации получается получить несмешенную оценку на градиент? Как выглядит эта оценка?

Для большой выборки можно на каждом шаге брать один случайный элемент или несколько элементов k . Если классы линейноразделимы, то точно знаем, что стохастический градиент является несмешенной оценкой градиента (есть там какая-то теорема), а значит метод сходится к локальному минимуму. На практике даже в случае линейноразделимых классов этот метод сходится. Сама формула примерно такая

$$\frac{1}{k} \sum_{i=1}^k \frac{f(x_i + h) - f(x_i - h)}{2h}$$

6. Что такое кросс-валидация? На что влияет количество блоков в кросс-валидации?

Разбиваем выборку на k частей, по очереди счиаем каждую часть тестом, а остальные тренируем, и смотрим на среднее полученных метрик качества. Чем меньше количество блоков, тем быстрее это работает, но тем менее мы уверены в отсутствии переобучения.

7. Чем гиперпараметры отличаются от параметров? Что является параметрами и гиперпараметрами в линейных моделях и в решающих деревьях? **Этот вопрос, вероятно, можно допилить, добавив забытые автором параметры и гиперпараметры этих моделей**

Параметры настраиваются непосредственно при обучении, в то время как гиперпараметры фиксированные и изменяются вручную, если мы понимаем, что модель учится плохо.

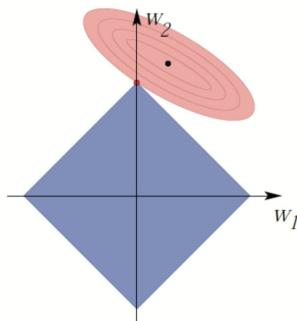
В линейных моделях гиперпараметром может служить вес регрессии в функции потерь, а параметры - матрица весов и вектор смещений. В решающих деревьях гиперпараметры: максимальная глубина, минимальное число элементов в листьях, а параметрами служат элементы разбиений: признаки и пороги.

8. Что такое регуляризация? Почему L1-регуляризация отбирает признаки?

Регуляризация - штраф за сложность модели, выражаящийся в дополнительных слагаемых в функции потерь, которые увеличивают функцию потерь при больших весах. Строгое объяснение почему L1 отбирает признаки довольно громоздкое, надеюсь, его не будут спрашивать. “Неубедительное” объяснение: линии уровня функции потерь как правило будут касаться ромба (области, которой L1 регуляризация ограничивает веса) в “углах”, то есть в точках, где одна или несколько координат равны 0.

Почему ℓ_1 разреживает: простое и неубедительно объяснение

$$\left\{ \begin{array}{l} \tilde{Q} = \sum_{i=1}^l L(M_i) \rightarrow \min \\ \sum_{k=1}^m |w_k| \leq \tau \end{array} \right.$$



9. Запишите формулу для линейной модели классификации. Что такое отступ?
10. Что такое точность и полнота?
Обозначения: TP, FP, TN, FN - true positive (мы ответили, что класс 1 (positive), и оказались правы (true)), false positive, true negative, false negative. $TPR = \frac{TP}{TP+FN}$, $FPR = \frac{FP}{FP+TN}$ - true positive rate, false positive rate.

$$Presision = \frac{TP}{TP+FP}$$
 — доля выстрелов, попавших в цель

$$Recall = \frac{TP}{TP+FN}$$
 — доля сбитых самолетов
11. Что такое ROC-AUC? Как построить ROC-кривую?
Пусть у нас есть некий бинарный классификатор, который умеет не просто выдавать классы объекта, а выдавать вероятность принадлежности классу 1. В обычной ситуации логично отнести объект к классу 1, если эта вероятность больше $\frac{1}{2}$. Можно же выбирать различные пороговые значения и получать различные результаты.
Итак, выбирая различные пороги, получаем различные точки $\left(\frac{FPR}{TPR}\right)$, по которым и строим кривую называемую ROC-кривой. ROC-AUC - это площадь под ней. Именно она и служит метрикой качества.
12. Запишите функционал логистической регрессии. Как он связан с методом максимума правдоподобия?
$$L = - \sum_{i=1}^n \sum_{j=1}^k y_i^j \ln \hat{y}_i^j$$
13. Запишите задачу метода опорных векторов для линейно неразделимого случая. Как функционал этой задачи связан с отступом классификатора? Найдите теорминимум. Впрочем ничего нового.
14. Опишите жадный алгоритм обучения решающего дерева.
15. Почему с помощью решающего дерева можно достичь нулевой ошибки на обучающей выборке без повторяющихся объектов?
16. Что такое бэггинг?
Есть выборка X и алгоритм $a_Y(x)$, который может, обучившись на некоторой выборке Y , давать ответ для объекта x . Делаем из выборки X много разных совокупностей X_1, X_2, \dots, X_n , которые создаем, случайно выбирая объекты из X по схеме с возвращением (то есть каждая из X_i - набор, быть может повторяющихся, случайно выбранных элементов X). И теперь нашим ответом для x будет усредненный по различным обучающим выборкам X_1, X_2, \dots, X_n ответ исходного алгоритма: $\frac{1}{n} \sum_{i=1}^n a_{X_i}(x)$.
17. Что такое случайный лес? Чем он отличается от бэггинга над решающими деревьями?
Случайный лес - это бэггинг над **рандомизированными** решающими деревьями. Рандомизированные деревья - это деревья, в которых мы в каждом узле выбираем лучшее разбиение не по всем признакам, а по случайному подмножеству признаков.

18. Как в градиентном бустинге обучаются базовые алгоритмы? Что такое сокращение шага?
19. Зачем нужен backprop, что такое производная вектора по вектору?
20. Чем нейросеть отличается от линейной модели, приведите примеры нейросетей?
21. Объясните идею weight sharing на примере сверточного слоя, почему эта техника работает именно с изображениями?
Какое свойство входных объектов мы учитываем?
22. В чем отличие между сверточными и рекуррентными слоями?
В рекуррентном слое на вход подается последовательность, и при обработке очередного её элемента учитывается так называемое скрытое состояние, которое учитывает какие объекты были поданы на вход до этого. Таким образом, такие слои помогают работать со структурами, в которых важна последовательность объектов. Например, с текстами.
23. Как работает метод K-Means?
Это метод решający задачу кластеризации в предположении известного числа кластеров. В начале рандомно задаем центры кластеров. Далее вычисляем к какому кластеру принадлежит каждый объект: к какому центру объект ближе - к тому кластеру его и относим. Затем пересчитываем центры кластеров: берем среднее по всем объектам, отнесенными к этому кластеру на предыдущем шаге. Повторяем предыдущие два шага, пока центры не стабилизируются.
24. Как работает метод t-SNE? Охуенный вопрос для теорминимума, конечно.
Эмбединг в пространство меньшей размерности с сохранением свойства локальности. Есть параметр перплексия, которая условно задает ожидаемое количество соседей.
25. Запишите постановку задачи в методе главных компонент.