

# ML 2017

3 курс ФИВТ, 2017

14 мая 2018 г.

## Содержание

<b>I Билеты</b>	<b>4</b>
<b>1 Билет 1</b>	<b>4</b>
1.1 Supervised и unsupervised learning . . . . .	4
1.1.1 Supervised learning . . . . .	4
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 . . . . .	5
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 . . . . .	6
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 . . . . .	7
1.6.2 Изображение . . . . .	7
1.6.3 Звук . . . . .	7
1.7 Предобработка признаков . . . . .	7
1.7.1 Разреженные признаки . . . . .	7
1.7.2 Категориальные признаки . . . . .	7
<b>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).</b>	<b>7</b>
2.1 Метрики качества в задачах классификации. . . . .	7
2.1.1 Accuracy . . . . .	7
2.1.2 Precision . . . . .	8
2.1.3 Recall . . . . .	8
2.1.4 F-мера . . . . .	8
2.1.5 ROC-AUC . . . . .	8
2.1.6 LogLoss . . . . .	9
2.2 Метрики качества в задачах регрессии. . . . .	9
2.2.1 MSE . . . . .	9
2.2.2 MAE . . . . .	9
2.2.3 MAPE . . . . .	9
2.2.4 SMAPE . . . . .	10

<b>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.</b>	<b>10</b>
3.1 Работа решающего дерева . . . . .	10
3.2 Как строятся решающие деревья? . . . . .	10
3.3 Как выдавать ответ в листе? . . . . .	11
3.4 Как выбрать наилучшее разбиение? . . . . .	11
3.5 Настройка гиперпараметров . . . . .	11
3.6 Преимущества и недостатки деревьев. . . . .	11
<b>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.</b>	<b>12</b>
4.1 Bagging = Bootstrap aggregation . . . . .	12
4.2 Stacking . . . . .	12
4.3 Blending . . . . .	12
4.4 Bias-variance trade-off . . . . .	12
<b>5 Бэггинг и Random Forest. Связь корреляции между ответами моделей и качеством модели в бэггинге.</b>	<b>13</b>
5.0.1 Бэггинг. . . . .	13
5.0.2 Random Forest. . . . .	13
<b>6 Бустинг и GBM. Выбор параметров в ансамблях решающих деревьев. Сравнение Random Forest и GBDT (подбор параметров, переобучение).</b>	<b>15</b>
6.1 Gradient boosting . . . . .	15
6.2 Обучение базовых алгоритмов в GB . . . . .	16
6.3 Описание алгоритма градиентного бустинга . . . . .	17
6.4 Градиентный бустинг для регрессии . . . . .	17
6.5 Градиентный бустинг для классификации . . . . .	17
6.6 Градиентный бустинг для решающих деревьев . . . . .	18
<b>7 Линейные модели в задачах классификации и регрессии: функции потерь, регуляризаторы, оптимизационные задачи. Стохастический градиентный спуск.</b>	<b>19</b>
7.1 Линейный классификатор . . . . .	19
7.2 Линейный регрессор . . . . .	19
7.3 Регуляризаторы . . . . .	19
<b>8 Линейная регрессия. Геометрический и аналитический вывод формулы для весов признаков. Регуляризация в линейной регрессии: гребневая регрессия и LASSO.</b>	<b>21</b>
8.1 Постановка задачи . . . . .	21
8.2 Вывод формулы для весов признаков . . . . .	22
8.3 Регуляризация в линейной регрессии . . . . .	22
<b>9 Логистическая регрессия. Варианты записи оптимизационной задачи. Оценка вероятности принадлежности к классу. Настройка параметров с помощью стохастического градиентного спуска.</b>	<b>23</b>
<b>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.</b>	<b>24</b>
<b>11 Нейронные сети, обучение (backprop), слои для нейронных сетей (dense, conv, pooling, batchnorm), нелинейности (relu vs sigmoid, softmax), функции потерь (logloss, l2, hinge)</b>	<b>26</b>
11.1 Обучение нейронных сетей . . . . .	26
11.2 Слои для нейронных сетей . . . . .	27
11.3 Нелинейности (relu vs sigmoid, softmax) . . . . .	29
11.4 Функции потерь (logloss, l2, hinge) . . . . .	31
<b>12 Нейронные сети, обучение (backprop), оптимизация для нейронных сетей (sg, msg, nmsg, rmsprop, adam), регуляризация нейросетей (dropout, dropconnect, l1, l2, batchnorm)</b>	<b>31</b>
12.1 Оптимизация: . . . . .	31
12.2 Регуляризация . . . . .	33

<b>13 Нейронные сети, обучение (backprop), современные сверточные нейронных сетей (vgg, resnet, inception) и детали обучения (batchnorm, pretraining)</b>	<b>33</b>
13.1 VGG . . . . .	33
13.2 Inception . . . . .	34
13.3 ResNet . . . . .	35
<b>14 Рекуррентные НС, обучение (backprop tt), отличие от сверточных, разновидности рекуррентных слоев (RNN, LSTM, GRU) аннотация изображений, перевод, диалоговые системы</b>	<b>35</b>
14.1 Рекуррентные НС . . . . .	35
14.2 Backprop ТТ . . . . .	36
14.3 Vanishing Gradient . . . . .	36
14.4 LSTM . . . . .	37
14.4.1 GRU . . . . .	39
14.5 Аннотация изображений, перевод, диалоговые системы . . . . .	39
<b>15 Задача снижения размерности пространства признаков. Идея метода главных компонент (PCA). Связь PCA и сингулярного разложения матрицы признаков (SVD). Идея методов SNE, tSNE, принципиальные отличия от PCA.</b>	<b>39</b>
15.1 Задача снижения размерности пространства признаков. Идея метода главных компонент (PCA) . . . . .	39
15.2 Идея методов SNE, tSNE, принципиальные отличия от PCA. . . . .	40
<b>16 Задача кластеризации. Агglomerативная и дивизионная кластеризация.</b>	<b>42</b>
<b>17 Кластеризация с помощью ЕМ-алгоритма (без вывода М-шага). Формула Ланса-Уилльямса.</b>	<b>44</b>
17.1 Кластеризация с помощью ЕМ-алгоритма (без вывода М-шага). Формула Ланса-Уилльямса. . . . .	44
17.2 Алгоритмы k-Means и DBSCAN. . . . .	45
<b>18 Теоретический минимум</b>	<b>47</b>
18.1 В чем разница между задачами классификации, кластеризации, регрессии, уменьшения размерности, приведите примеры. . . . .	47
18.2 Что такое объект, целевая переменная, признак, модель, функционал ошибки и обучение? . . . . .	47
18.3 Запишите формулы для линейной модели регрессии и для среднеквадратичной ошибки. . . . .	48
18.4 Что такое градиент? Какое его свойство используется при минимизации функций? . . . . .	48
18.5 Запишите формулу для одного шага градиентного спуска. Как модифицировать градиентный спуск для очень большой выборки? . . . . .	48
18.6 Что такое кросс-валидация? На что влияет количество блоков в кросс-валидации? . . . . .	48
18.7 Чем гиперпараметры отличаются от параметров? Что являются параметрами и гиперпараметрами в линейных моделях и в решающих деревьях? Этот вопрос, вероятно, можно допилить, добавив забытые автором параметры и гиперпараметры этих моделей . . . . .	49
18.8 Что такое регуляризация? Чем на практике отличается L1-регуляризация от L2? . . . . .	49
18.9 Запишите формулу для линейной модели классификации. Что такое отступ? . . . . .	49
18.10Что такое точность и полнота? . . . . .	49
18.11Что такое ROC-AUC? Как построить ROC-кривую? . . . . .	49
18.12Запишите функционал логистической регрессии. Как он связан с методом максимума правдоподобия? . . . . .	50
18.13Запишите задачу метода опорных векторов для линейно неразделимого случая. Как функционал этой задачи связан с отступом классификатора? . . . . .	50
18.14Опишите жадный алгоритм обучения решающего дерева. . . . .	50
18.15Почему с помощью решающего дерева можно достичь нулевой ошибки на обучающей выборке без повторяющихся объектов? . . . . .	50
18.16Что такое бэггинг?	51
18.17Что такое случайный лес? Чем он отличается от бэггинга над решающими деревьями?	51
18.18Как в градиентном бустинге обучаются базовые алгоритмы? Что такое сокращение шага?	51
18.19Зачем нужен backprop, что такое производная вектора по вектору?	51
18.20Чем нейросеть отличается от линейной модели, приведите примеры нейросетей?	51
18.21Объясните идею weight sharing на примере сверточного слоя, почему эта техника работает именно с изображениями? Какое свойство входных объектов мы учитываем?	51
18.22В чем отличие между сверточными и рекуррентными слоями?	51
18.23Как работает метод K-Means?	51
18.24Как работает метод t-SNE?	52
18.25Запишите постановку задачи в методе главных компонент.	52

# 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 scoring — 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 туда.

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  $\rho$ , 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).  $\mu_y$  — 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 тренировочной выборки (кросс-валидация). Если кривые почти совпадают и расходятся, значит недообучение. Если кривая тестовой выборки уходит вниз от кривой тренировочной 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.

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

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

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

### 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 Раэрженные 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:  $accuracy = \frac{5+90}{5+90+10+5} = 86.4$ .

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

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

## 2.1.3 Recall

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

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

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.

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

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).

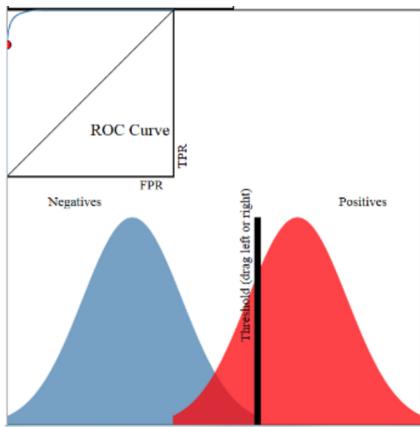
ROC-кривая - зависимость TPR (True Positive Rate) от FPR (False Positive Rate).

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

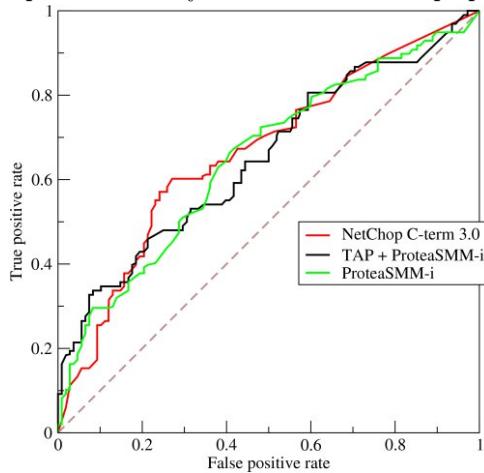
$$FPR = \frac{FP}{FP + TN}.$$

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:



## 2.1.6 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 Метрики качества в задачах регрессии.

### 2.2.1 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.

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

### 2.2.3 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.

### 2.2.4 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?

1. Строим разбиение выборки по значению одного из признаков. ("вот этот признак меньше какого-то порога"). Признак и порог нужно выбрать "наилучшим образом".
2. Выборка делится по этому условию на две части.
3. В каждой из них тоже можно сделать наилучшее разбиение.
4. 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 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

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

### 3.6 Преимущества и недостатки деревьев.

Достоинства:

- Легко восстанавливают сложные зависимости
- Просты для понимания и интерпретации
- Не требуют нормализации и других способов предобработки данных
- 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  $L(x, D) = \text{Learn}(x, D)$   $T(x) = \text{Truth}(x)$

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

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

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

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

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

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

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

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

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

Посчитаем 2-ое слагаемое:

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

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

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

То есть в конечном итоге мы получаем:

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

$=$

## 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:

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

Задача из домашки, которая может помочь:

Покажите, что если есть  $M$  одинаково распределенных случайных величин с дисперсией  $\sigma^2$ , любые две из которых имеют положительную корреляцию  $\rho$ , то дисперсия их среднего будет равна:

$$\rho\sigma^2 + (1 - \rho)\frac{\sigma^2}{M}$$

$$D(\bar{X}) = \frac{1}{M^2} cov(\sum X_i, \sum X_i) = \frac{1}{M^2} \left[ \sum_{i,j} cov(X_i, X_j) \right] = \\ \frac{1}{M^2} (M\sigma^2 + M(M-1)\rho\sigma^2) = \frac{\sigma^2}{M} + \frac{(M-1)\rho\sigma^2}{M} = \rho\sigma^2 + (1 - \rho)\frac{\sigma^2}{M}$$

Что и требовалось.

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 независимыми получаются базовые алгоритмы.)

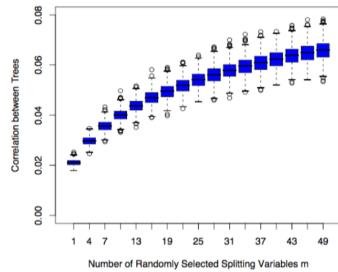
### 5.0.1 Бэггинг.

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:

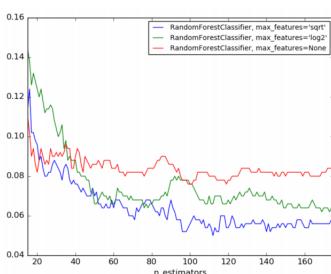
- В задаче регрессии имеет смысл брать  $q = \frac{d}{3}$ , то есть использовать треть от общего числа признаков.
- В задаче классификации имеет смысл брать  $q = \sqrt{d}$

### 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 случайного подмножества размера  $q$ .
  - Следует обратить особое внимание, что случайное подмножество 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 больше качество большой модели - бэггинга. (Но это при условии одинакового качества маленьких моделей)

Проблема Random Forest в том, что эё 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, ...).

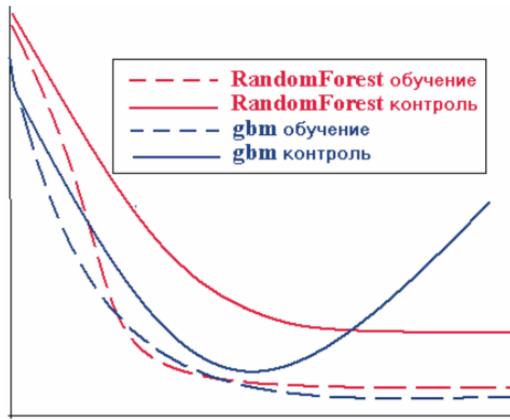
В random forest переобучения добиться довольно сложно, поэтому количество деревьев ограничивается из соображений времени работы. Кроме того при большом количестве деревьев начинают появляться похожие, а скоррелированные деревья пользы не приносят, так что делать их количество слишком бессмысленно.

Сравнение:

- RF почти не требует настройки. Один параметр: количество признаков, которые перебираются в node.
- GB в среднем работает лучше
- RF можно параллелить, GB определенно последовательный
- RF не так сильно переобучается.
- RF строится из полноценных деревьев, у которых большой variance, затем собирает их в композицию, уменьшая variance. GB строится из низких деревьев, может даже, пней, у которых, соответственно, высоких bias, затем собирает их так, чтобы уменьшить bias.

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  $R_i$ ,  $i = 1, \dots, J$  – 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:

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

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.

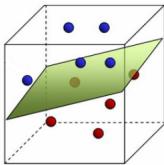
### 7.1 Линейный классификатор

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  $a(x) = \text{sign}\{f(x)\}$  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:

$$\begin{aligned} Q(M) &= (1 - M)^2 \\ V(M) &= (1 - M)_+ \\ S(M) &= 2(1 + e^M)^{-1} \\ L(M) &= \log_2(1 + e^{-M}) \\ E(M) &= e^{-M} \end{aligned}$$

### 7.2 Линейный регрессор

В целом тоже, что и линейный классификатор, только  $f(x) = \langle w, x \rangle + w_0$  и будет целевой функцией без всякого сигнума.  
Задача оптимизации:

$$\sum_{i=1}^n L(y_i, f(x_i)) \rightarrow \min_w$$

Чаще всего в качестве  $L$  берут квадратичную функцию потерь.

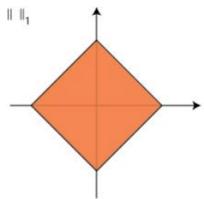
### 7.3 Регуляризаторы

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

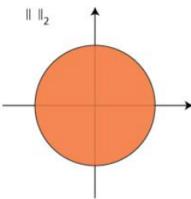
# Регуляризация

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

*l1 – регуляризация*



*l2 – регуляризация*

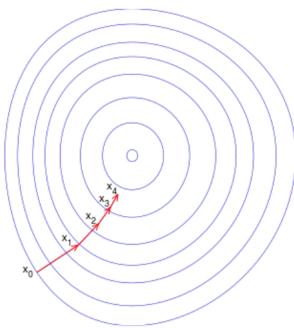


## Стохастический градиентный спуск.

Суть градиентного спуска - минимизация функции с помощью последовательных небольших шагов в сторону анти-градиента (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<sub>i</sub> – случайный элемент обучающей выборки*

Если я правильно понимаю, γ<sub>n</sub> - 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.

### 8.1 Постановка задачи

Линейная регрессия

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

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

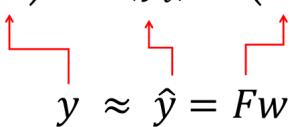
$$L(y_i, a(x_i)) = (y_i - a(x_i))^2$$

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

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

Модель и матричная запись

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


  
 $y \approx \widehat{y} = Fw$

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

$y$  - реальные ответы,  $\widehat{y}$  - ответы алгоритма,  $F$  - матрица признаков,  $w$  - веса признаков.

## 8.2 Вывод формулы для весов признаков

Аналитический вывод

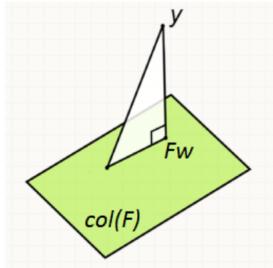
$$\frac{\partial(y - Fw)^2}{\partial w} = 2F^T(y - Fw) = 0$$

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

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

Геометрическая интерпретация

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



$$\begin{aligned} F_{(k)}^T (Fw - y) &= 0 \quad \forall k \\ F^T (Fw - y) &= 0 \\ F^T F w &= F^T y \end{aligned}$$

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

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

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

## 8.3 Регуляризация в линейной регрессии

LASSO - l1-регуляризация для линейной классификации, гребневая регрессия (Ridge) - l2-регуляризация. LASSO обнуляет веса, за счет чего можно отбирать признаки. Строгое доказательство весьма нетривиально и спрашивать его не должны, посмотреть его можно [здесь](#) начиная с 10:35. А если кому-то интересно, как его решать (а такие судя по диалогу в табличке есть. Хотя почему-то нагуглить этот кто-то сам и вставить это сюда не способен), то это можно прочитать [здесь](#). Но это нафиг не нужно знать. Далее поговорим про свойства гребневой регрессии.

## Гребневая регрессия ( $\ell_2$ -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  $\gamma$  на диагонали, за счет чего увеличиваем все собственные числа, а значит и улучшаем обусловленность. Так же есть теорема, что существует  $\gamma$  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  $\gamma$  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

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

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

Как правило, добавляется  $\ell_1$  или  $\ell_2$ -регуляризация, а оптимизационная задача решается с помощью SGD или метода Ньютона-Рафсона

(Из Воронцова:) Метод логистической регрессии основан на довольно сильных вероятностных предположениях, которые 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.

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

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

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

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

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

Фактически, это распределение бернулли с параметром  $f(< w, x >)$ .

### **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

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

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

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

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)

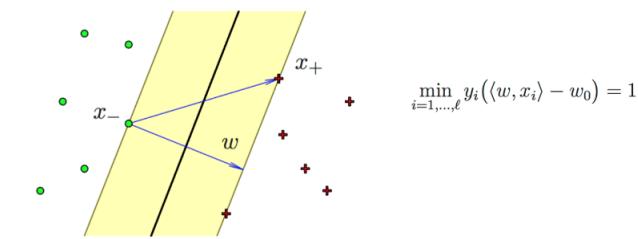
## 10 Метод опорных векторов: оптимизационная задача в условной и безусловной форме. Опорные векторы (достаточно записать и продифференцировать Лагранжиан, оптимизационная задача, выраженная через двойственные переменные - опционально). Идея Kernel Trick.

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 -  $\xi_i$ .  $y_i(\langle w, x_i \rangle - w_0) \geq 1 - \xi_i$  - добавление ошибки. Сумму добавляем, чтобы ошибки все-таки не были большими. А условие  $\geq 0$ , вроде, очевидно... Ура! Условная задача SVM!

## Случай линейно неразделимой выборки

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

Теперь к безусловной. Возьмем и перепишем все уравнения на  $\xi_i$  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

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

Опорные вектора: В общем, берем мы такие 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? Же сделаем неявно - зададим просто новое скалярное произведение.  $K(w, x) = \langle \phi(w), \phi(x) \rangle$ ,  $\phi$  - 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:

1. **Input Layer** Начальный слой, который служит для того, чтобы пропустить через сеть очередную порцию данных. У него нет параметров, количество нейронов в нём фиксированно и не является гиперпараметром.
2. **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).
3. **Output Layer** Выходной слой, количество нейронов в котором фиксированно, не содержит параметров. Обычно используется для подсчета лосса.

## 11.1 Обучение нейронных сетей

Как обучаются нейронные сети?

Нейронная сеть обучаётся 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.

**BackPropagation** это метод вычисления градиента лосс-функции по параметрам.

[Лекция, разжевывающая backprop.](#)

Нейросеть - это некая (может быть, очень-очень большая и громоздкая) функция  $f(X, W_1, \dots)$ . Она представима как композиция некоторых функций, поэтому можно построить график вычислений (обычно он строится топологической сортировкой по функциям).

Рассмотрим картинку:

Backpropagation: a simple example

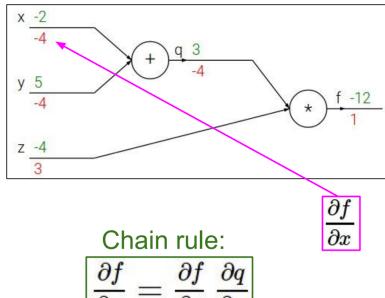
$$f(x, y, z) = (x + y)z$$

e.g.  $x = -2, y = 5, z = -4$

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

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

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



Chain rule:

$$\frac{\partial f}{\partial x} = \frac{\partial f}{\partial q} \frac{\partial q}{\partial x}$$

На картинке у нас функция  $f$ , сначала она разбивается на график 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.

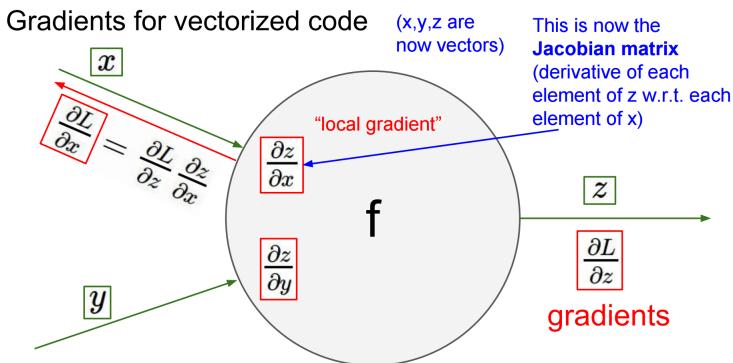
Рассмотрим самую правую вершину, у нас есть 2 входа –  $x + y$ , который мы обозначим  $q$ , и  $z$ .

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.

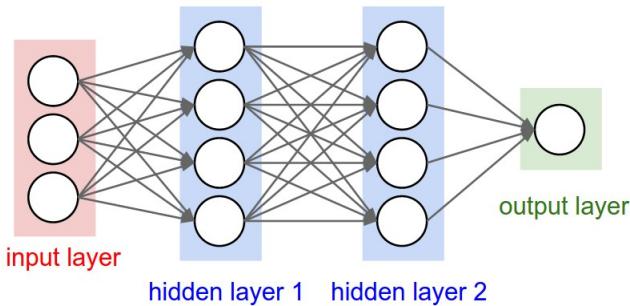
На фазе forward-pass (когда вы пропускаете данные через сеть, умножая на матрицы и прочее), вычисляется выход вершинки  $z$ , который определяется через какую-то функцию, затем вычисляются производные выхода по выходам, т.е.  $\frac{\partial z}{\partial x}$  и  $\frac{\partial z}{\partial y}$ , т.к.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

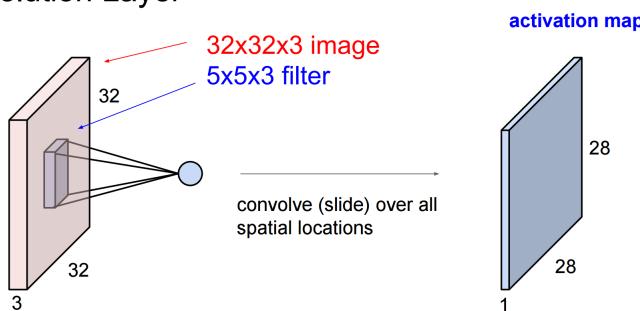


Dense Layer или Fully Connected Layer – слой, состоящий из  $n$  нейронов. Пусть предыдущий слой имел  $k$  выходов, тогда к каждому нейрону в FC слое идет  $k$  взвешенных рёбер, после этого (обычно) берётся их взвешенная сумма и применяется нелинейность.

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:

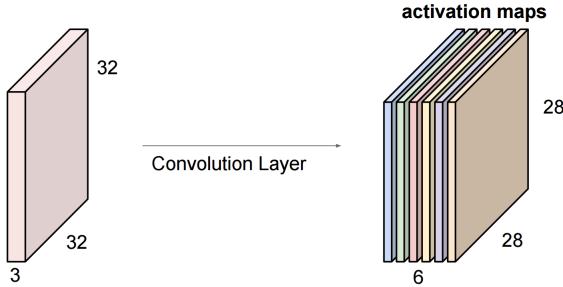
Convolution Layer



Convolutional Layer или сверточный слой. Идея в том, чтобы анализировать части наших данных независимо, поэтому вводится понятие фильтра. На картинке фильтр размера  $5 \times 5$ , 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  $h \times h$ ). Между частями картинки и фильтром берется 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  $5 \times 5$  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:

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

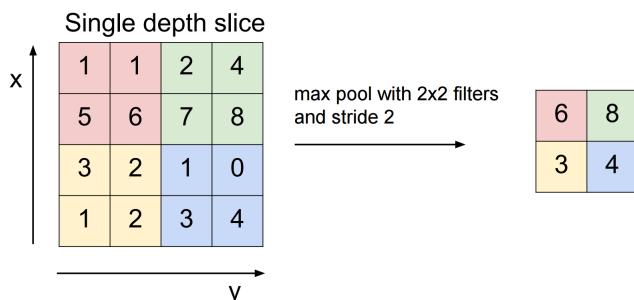
Важное следствие, если оставить делать свертки с  $S = 1$ ,  $P = 1$ , то размерность ваших Activation Map будет снижаться. Мы этого не хотим. Тогда мы можем сделать свертки с  $S = 1$ ,  $P = 2$ , проверяя формулу, мы увидим, что размерность тогда останется такой же, это хороший хак.

Насчет сверток  $1 \times 1$  – иногда это круто, они используются, чтобы снизить размерность тензора, а также для перемешивания. Напоминание: размерность глубины = количеству фильтров. Также свертка  $1 \times 1$  представляет собой протаскивание Fully Connected слоя через вашу картинку (т.к. веса одни и те же).

### 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'a 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:  $\gamma, \beta$

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

$$\begin{aligned} \mu_{\mathcal{B}} &\leftarrow \frac{1}{m} \sum_{i=1}^m x_i && // \text{mini-batch mean} \\ \sigma_{\mathcal{B}}^2 &\leftarrow \frac{1}{m} \sum_{i=1}^m (x_i - \mu_{\mathcal{B}})^2 && // \text{mini-batch variance} \\ \hat{x}_i &\leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} && // \text{normalize} \\ y_i &\leftarrow \gamma \hat{x}_i + \beta \equiv BN_{\gamma, \beta}(x_i) && // \text{scale and shift} \end{aligned}$$

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 (пояснить это я, конечно, не буду, вот [тут](#) подробнее).

### 11.3 Нелинейности (relu vs sigmoid, softmax)

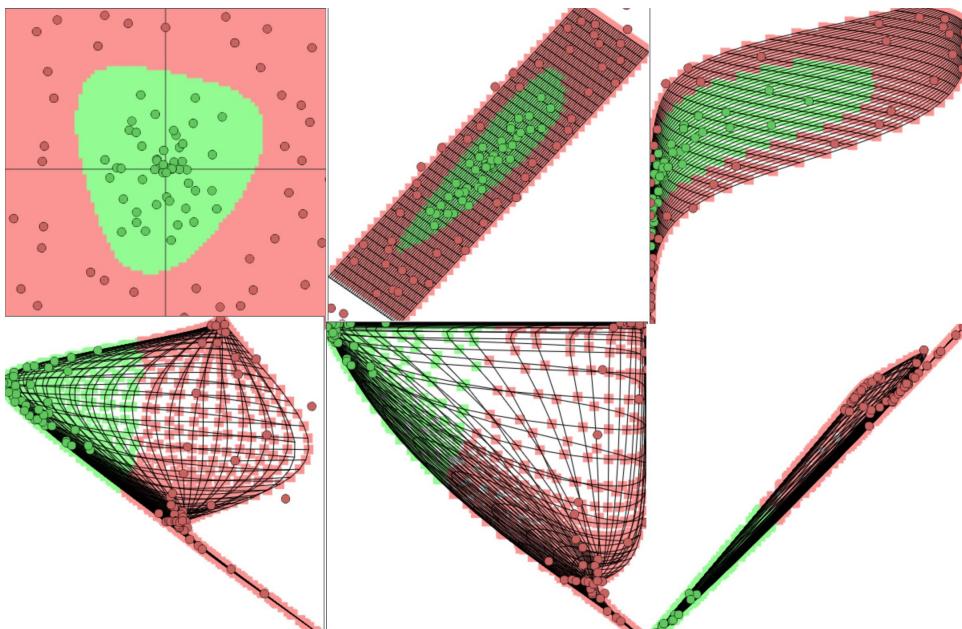
Когда мы умножаем на матрицу (FC layer), мы совершаём афинное преобразования пространства, но это не слишком круто, поэтому давайте введём нелинейности.

Фишкa 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.

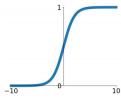
На первой картинке мы видим наши данные. Дальше первый FC просто аффинно изменяет, а вот тангенс уже прикольно искривляет наше пространство на 3 картинке. Ещё 2 картинки пытаются как-то изменить пространство и в итоге в конце мы получаем линейноразделимую плоскость, которую наша сеть-классификатор легко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

Перейдем к описанию нелинейностей:

## Activation Functions

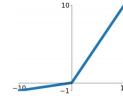
### Sigmoid

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



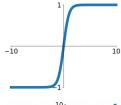
### Leaky ReLU

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



### tanh

$$\tanh(x)$$

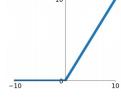


### Maxout

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

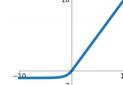
### ReLU

$$\max(0, x)$$



### 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. Из-за константного градиента быстро считается. Прореживает матрицу – здорово! +

## 5. Softmax

Делает из вектора длины  $k$  вектор длины  $k$ . Аналог многомерной логистической функции. Выходы - типа вероятность принадлежности к классам. Обычно ставится в конце. При очень больших значениях  $x$  градиент может неконтролируемо уменьшиться.

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

## 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.

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

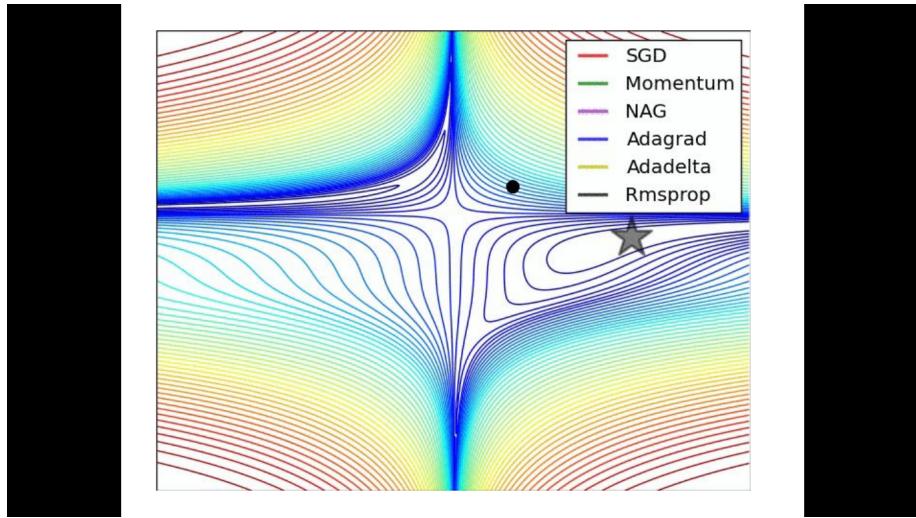
где  $\hat{Y}$  - предсказанный результат,  $Y$  реальный.

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

$N$  – 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, nmmsg, 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.  $\rho$  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

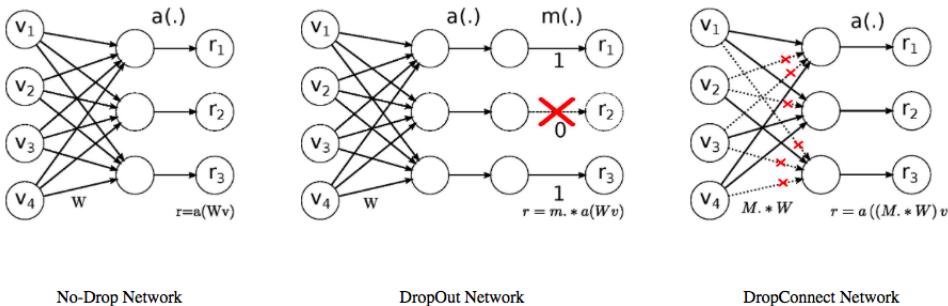
Adam with beta1 = 0.9,  
beta2 = 0.999, and learning\_rate = 1e-3 or 5e-4  
is a great starting point for many models!

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 случиться 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.



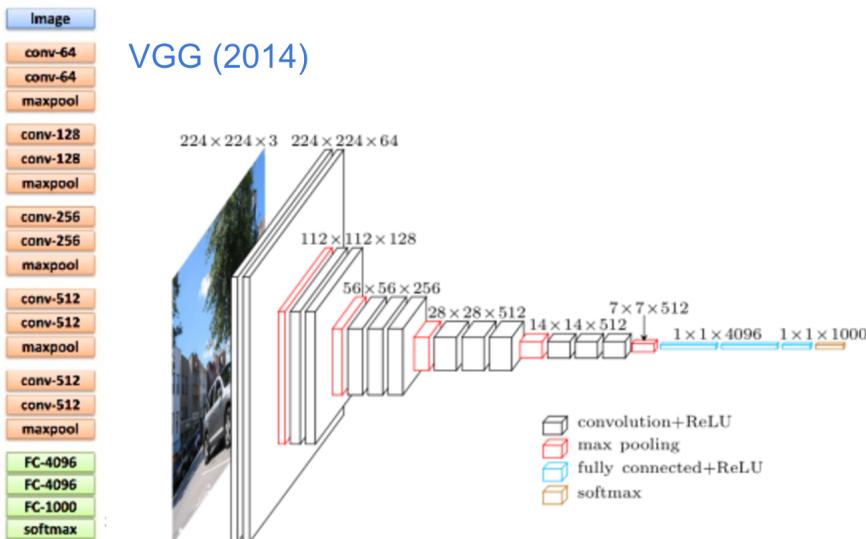
- **l1, l2** - функции сопоставляющие одному или нескольким слоям штраф за большие значения весов (вычисляются по соответствующим формулам:  $\lambda \sum_i |w_i|$ ,  $\lambda \sum_i |w_i|^2$ ), для минимизации добавляются к общему лоссу сети (большие коэффициенты сети также можно считать переобучением)

- **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)

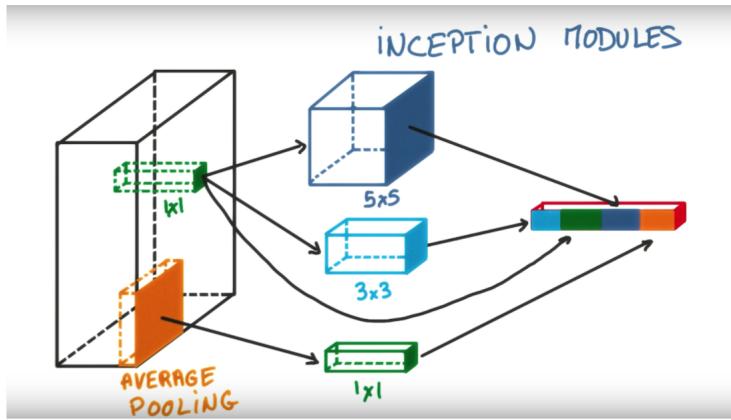
### 13.1 VGG



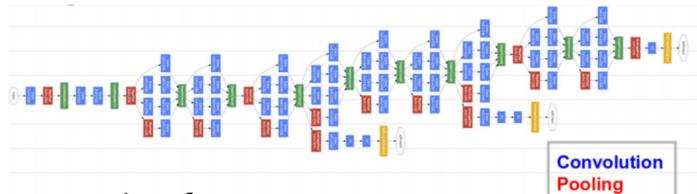
VGG - свёрточная сеть из 2014 года, основанная на куче свёрток  $3 \times 3$ , а затем пулингов. Её изобрели ещё **до** появления BatchNorm, поэтому с обучением были сложности. Самой глубокой сетью была VGG-19, но обучить её сходу не получалось, поэтому сначала обучали сеть гораздо меньшего размера, 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 сетьми.

## 13.2 Inception

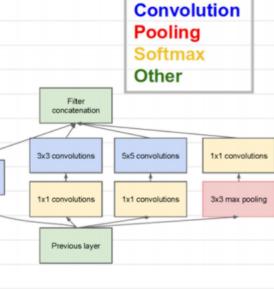


GoogleNet (2014)



"Large but fast":

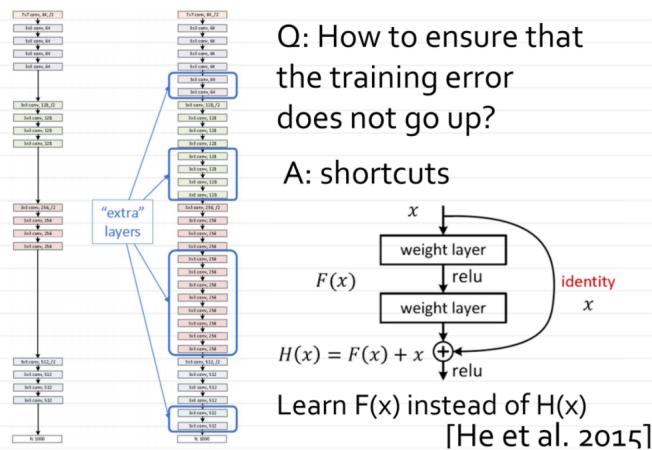
- Keep large number of maps
- apply convolutions only to dimensionality reduced stacks



Inception - свёрточная сеть от гугла, вышла следом за VGG. Люди подумали, почему мы делаем только  $3 \times 3$  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

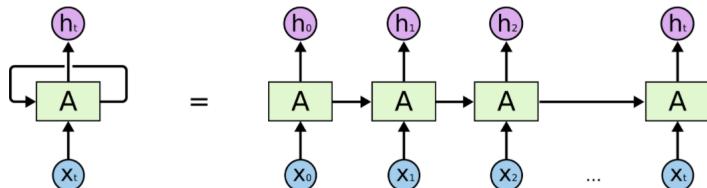
#### ResNet (2015)



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  $y_i$ . Таким образом они могут принимать на вход последовательности из нескольких объектов. Например, тексты.

Формула для очередного состояния:

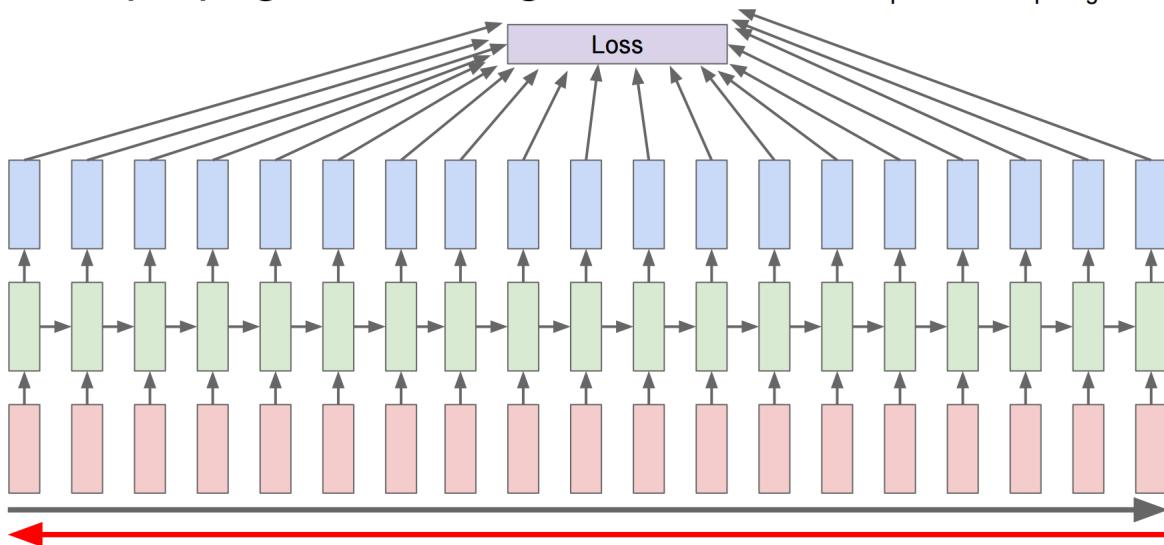
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t), \quad y_t = W_{hy}h_t.$$

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.

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

где  $\beta$  - верхняя граница нормы.

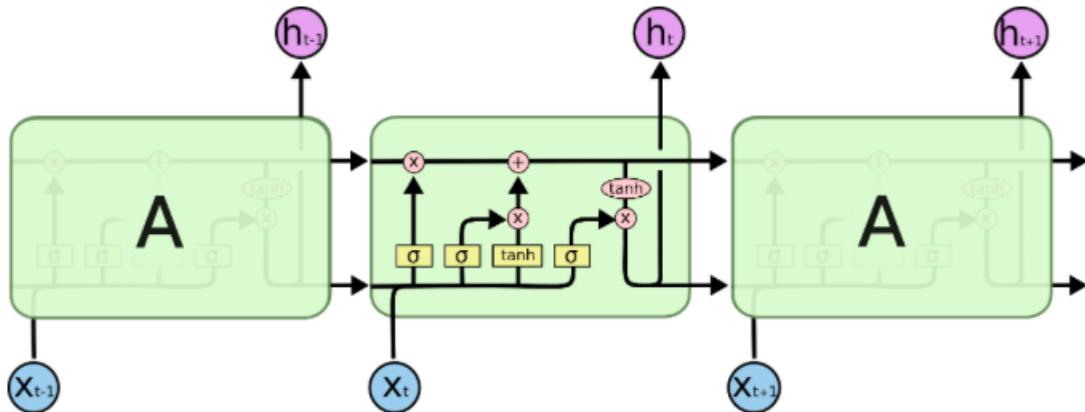
Тогда

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

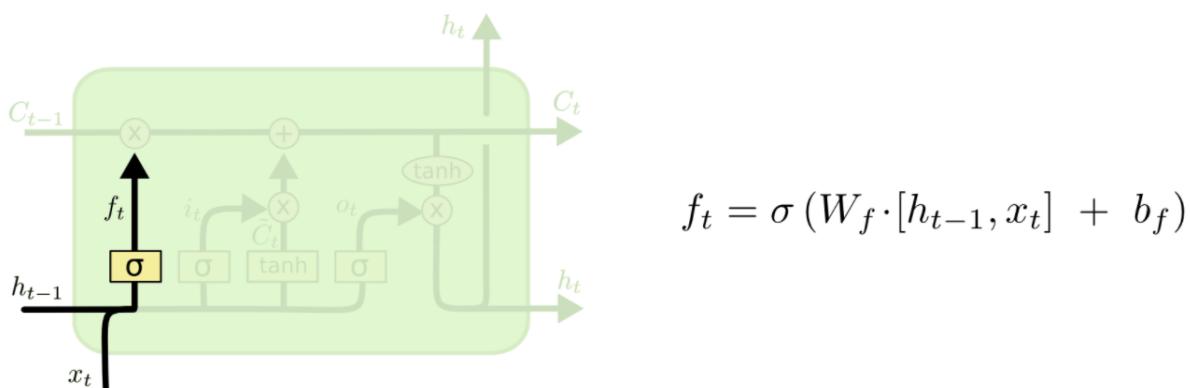
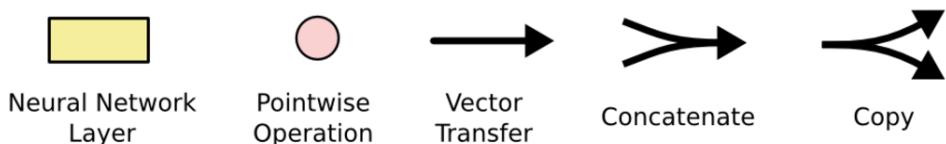
Функция экспоненциальна от длины последовательности, да и экспонента, поэтому в Backprop TT очень часто затухает, или наоборот - взрывается, градиент.

Обычно в vanilla RNN ограничивают длину последовательности, по которой делают BPRTT. Проблемы, связанные с взрывающимся градиентом, решить проще, чем с vanishing gradient. Во-первых, взрыв градиента проще обнаружить: веса уходят на бесконечность и сеть выдает NaN-ы. Для предотвращения этого делают gradient clipping - ограничивают градиент, если он выше какого-то значения. Однако для борьбы с затухающими градиентами таких методов нет. Поэтому используют модификацию RNN, а именно LSTM.

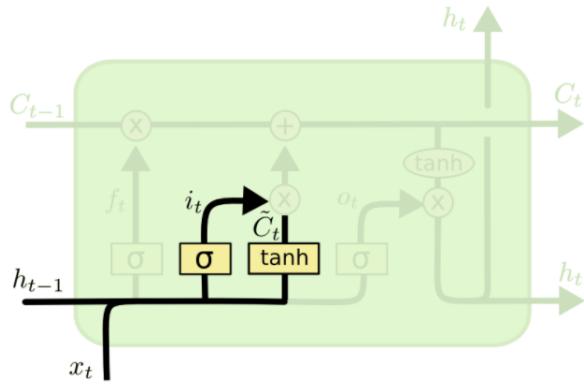
## 14.4 LSTM



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".



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.

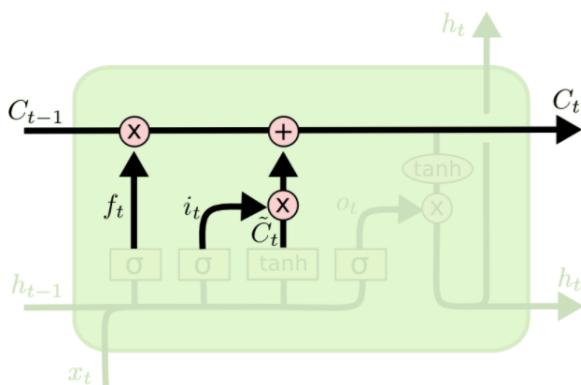


$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

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

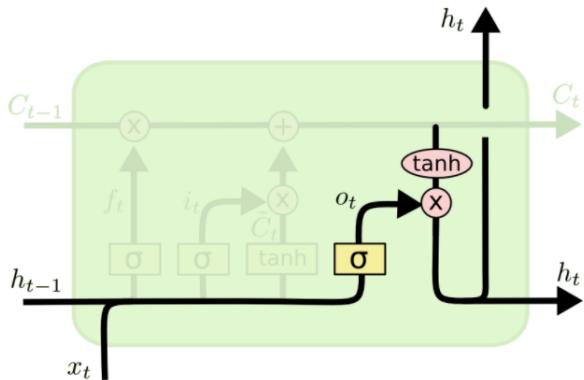
Далее есть  $i_t$  - input layer, и  $\tilde{C}_t$  - candidate layer. Input Layer решает, какие значения мы обновим, а какие оставим. А Candidate layer решает, какие значения нам нужно будет добавить в  $c_t$  - cell state.

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.



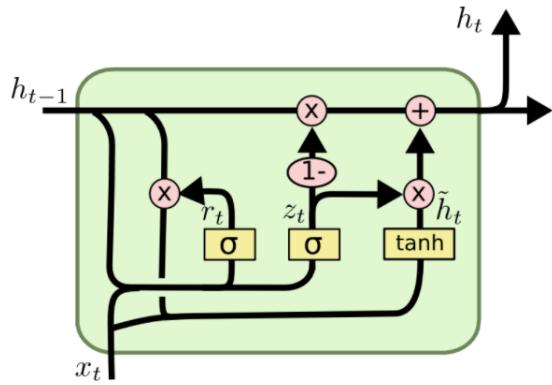
$$o_t = \sigma(W_o [h_{t-1}, x_t] + b_o)$$

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

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.

#### 14.4.1 GRU



$$z_t = \sigma (W_z \cdot [h_{t-1}, x_t])$$

$$r_t = \sigma (W_r \cdot [h_{t-1}, x_t])$$

$$\tilde{h}_t = \tanh (W \cdot [r_t * h_{t-1}, x_t])$$

$$h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$$

"Упрощенный" LSTM. Надо смотреть на картинку.

Для начала поймем, что такое "1"-посередине картинки. Как видно из формулы,  $h_t = (1 - z_t) * h_{t-1} + z_t * \tilde{h}_t$ . Здесь  $z_t$  выполняет роль *input-gate* в LSTM, а  $1 - z_t$  — роль *forget-gate* в LSTM. То есть мы обновляем только то, что забыли. Логично правда?

$r_t$  — это аналог, виличимо, *output-gate*, то есть сколько из старого состояния мы протащим в обновление/ответ.

Как можно заметить, тут нет различий между  $c_t$  и  $h_t$ , а так же что мы прорасываем на следующий слой то же самое, что выдаем на выход.

Зачем все это нужно? У нас теперь меньше параметров, поэтому работает быстрее, и, как показывают тесты, не сильно хуже.

#### 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 методов SNE, tSNE, принципиальные отличия от PCA.

#### 15.1 Задача снижения размерности пространства признаков. Идея метода главных компонент (PCA).

**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.

Другая интерпретация:

В методе главных компонент строится минимальное число новых признаков, по которым исходные признаки восстанавливаются линейным преобразованием с минимальными погрешностями. PCA относится к методам обучения без учителя (unsupervised learning), поскольку матрица «объекты–признаки»  $F$  преобразуется без учёта целевого вектора  $y$ . Важно отметить, что PCA подходит и для регрессии, и для классификации, и для многих других типов задач анализа данных, как вспомогательное преобразование, позволяющее определить эффективную размерность исходных данных.

**Более подробно:**

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

$X = U\Sigma V^T$ , где  $\Sigma$  — 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 сопряженно-транспонированная матрица к  $V$ ).

Кроме того в  $\Sigma$  сингулярные числа идут по убыванию ( $\lambda_1 > \dots > \lambda_{\min(n,m)}$ ). 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 все, 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  $\mathbf{0}$  ввиду допущения D1), а также углы между всевозможными парами прямых, соединяющих объекты с  $\mathbf{0}$ .

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

## 15.2 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. Честно говоря, интерпретация с пружинками неочень, имхо. Я бы сказал так: в SNE мы задаем меру близости точек в многомерном пространстве и меру близости точек в пространстве меньшей размерности. А дальше подгоняем образы точек выборки так, чтобы эти меры были как можно более похожими. [Про SNE и tSNE](#)

[Про PCA](#) (с 89)

Начнем с SNE:

Задача: набор точек в пр-ве существенно больше 3, хотим переменную в пр-ве размерности 2-3, т.ч она максимально сохраняла бы структуру и закономерности исходных данных.

SNE начинается с преобразования многомерной евклидовой дистанции между точками в условные вероятности, отражающие сходство точек:

$$p_{j|i} = \frac{\exp\left(-\frac{\|x_i - x_j\|^2}{2\sigma_i^2}\right)}{\sum_{k \neq i} \exp\left(-\frac{\|x_i - x_k\|^2}{2\sigma_i^2}\right)}.$$

Т.е. насколько точка  $X_j$  близка к точке  $X_i$  при гауссовом распределении вокруг  $X_i$  с заданным отклонением  $\sigma$ . Сигма будет различной для каждой точки. Она выбирается так, чтобы точки в областях с большей плотностью имели меньшую дисперсию. Для этого используется оценка перплексии:

$$Perp(P_i) = 2^{H(P_i)}.$$

Где  $H(P_i)$  – энтропия Шеннона в битах

$$H(P_i) = - \sum_j p_{j|i} \log_2 p_{j|i}.$$

Перплексия - сглаженная оценка эффективного количества «соседей» для точки  $X_i$ . Авторы рекомендуют использовать значение в интервале от 5 до 50.

Для двумерных или трехмерных «коллег» пары  $X_i$  и  $X_j$ , назовем их для ясности  $Y_i$  и  $Y_j$ , условную вероятность оценим также через формулу 1. Стандартное отклонение  $\frac{1}{\sqrt{2}}$ :

$$q_{j|i} = \frac{\exp(-\|y_i - y_j\|^2)}{\sum_{k \neq i} \exp(-\|y_i - y_k\|^2)}.$$

Если точки отображения  $Y_i$  и  $Y_j$  корректно моделируют сходство между исходными точками высокой размерности  $X_i$  и  $X_j$ , то соответствующие условные вероятности  $p_{j|i}$  и  $q_{j|i}$  будут эквивалентны. Функция потерь для данного метода будет определяться формулой 3:

$$Cost = \sum_i KL(P_i || Q_i) = \sum_i \sum_j p_{j|i} \log \frac{p_{j|i}}{q_{j|i}}.$$

При этом градиент выглядит на удивление просто:

$$\frac{\partial Cost}{\partial y_i} = 2 \sum_j (p_{j|i} - q_{j|i} + p_{i|j} - q_{i|j})(y_i - y_j).$$

Поиск равновесия предлагается делать с учетом моментов:

$$Y^{(t)} = Y^{(t-1)} + \tau \frac{\partial Cost}{\partial Y} + \alpha(t)(Y^{(t-1)} - Y^{(t-2)}),$$

Использование классического SNE позволяет получить неплохие результаты, но может быть связано с трудностями в оптимизации функции потерь и проблемой скученности (в оригиналe – crowding problem). t-SNE если и не решает эти проблемы совсем, то существенно облегчает. Функция потерь t-SNE имеет два принципиальных отличия. Во-первых, у t-SNE симметричная форма сходства в многомерном пространстве и более простой вариант градиента. Во-вторых, вместо гауссова распределения для точек из пространства отображения используется t-распределение (Стьюдента), «тяжелые» хвосты которого облегчают оптимизацию и решают проблему скученности.

#### Теперь про t-SNE:

Предлагается минимизировать одиночную дивергенцию между совместной вероятностью  $P$  в многомерном пространстве и совместной вероятностью  $Q$  в пространстве отображения

$$Cost = KL(P || Q) = \sum_i \sum_j p_{ij} \log \frac{p_{ij}}{q_{ij}},$$

где  $p_{ii}$  и  $q_{ii} = 0$ ,  $p_{ij} = p_{ji}$ ,  $q_{ij} = q_{ji}$  для любых  $i$  и  $j$ , а  $p_{ij}$  определяется по формуле:

$$p_{ij} = \frac{p_{i|j} + p_{j|i}}{2n},$$

где  $n$  — количество точек в наборе данных. Градиент для симметричного SNE получается существенно проще, чем для классического:

$$\frac{\partial Cost}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j).$$

Проблема скученности заключается в том, что расстояние между двумя точками в пространстве отображения, соответствующими двум среднедаленым точкам в многомерном пространстве, должно быть существенно больше, нежели расстояние, которое позволяет получить гауссово распределение. Проблему решают хвосты Стьюдента. В t-SNE используется t-распределение с одной степенью свободы. Совместная вероятность для пространства отображения в этом случае будет определяться формулой 4:

$$q_{ij} = \frac{(1 + \|y_i - y_j\|^2)^{-1}}{\sum_{k \neq l} (1 + \|y_k - y_l\|^2)^{-1}}.$$

А соответствующий градиент – выражением 5:

$$\frac{\partial Cost}{\partial y_i} = 4 \sum_j (p_{ij} - q_{ij})(y_i - y_j)(1 + \|y_i - y_j\|^2)^{-1}.$$

Возвращаясь к физической аналогии, результирующая сила, определяемая формулой 5, будет существенным образом стягивать точки пространства отображения для близлежащих точек многомерного пространства, и отталкивать — для удаленных.

**Алгоритм t-SNE** в упрощенном виде можно представить следующим псевдокодом:

Data: набор данных  $X = \{x_1, x_2, \dots, x_\tau\}$ ,

параметр функции потерь: перплексия  $Perp$ ,

Параметры оптимизации: количество итераций  $T$ , скорость обучения  $\tau$ , момент  $\alpha(t)$ .

Result: представление данных  $Y(T) = \{y_1, y_2, \dots, y_n\}$  (в 2D или 3D).

1. вычислить попарное сходство  $p_{j|i}$  с перплексией  $Perp$  (используя формулу 1)
2. установить  $p_{ij} = \frac{(p_{j|i} + p_{i|j})}{2n}$ .
3. инициализировать  $Y(0) = \{y_1, y_2, \dots, y_n\}$  точками нормального распределения (mean=0, sd=1e-4)
4. for  $t = 1$  to  $T$  do
  - (a) вычислить сходство точек в пространстве отображения  $q_{ij}$  (по формуле 4)
  - (b) вычислить градиент  $\frac{\partial Cost}{\partial y}$  (по формуле 5)
  - (c) установить  $Y(t) = Y(t-1) + \tau \frac{\partial Cost}{\partial y} + \alpha(t)(Y(t-1) - Y(t-2))$ .

Принципиальные отличия:

1. РСА линеен (в связи с чем может меньше), SNE и tSNE нет
2. SNE и tSNE неинтерпретируемы, а в PCA первые несколько главных компонент как правило можно интерпретировать
3. tSNE недетерминирован, от запуска к запуску результат может существенно меняться, а PCA детерминирован.
4. PCA может сжимать размерность не до 2-3, а до произвольного числа. tSNE круто работает в основном только для сжатия в размерность 2-3.
5. Эти алгоритмы можно успешно комбинировать. Сначала применяем PCA, отобрав признаки, откинув совсем уж неинформативные. А затем к полученному применяем tSNE.
6. Добавьте, чего тут нет

## 16 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  $\rho$ , 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.

Алгоритм агglomerативной кластеризации:

Пусть  $U, V$  – кластера, тогда обозначим за  $R(U, V)$  расстояние между этими кластерами (расстояние можно определять по-разному, подробности ниже)

Тогда алгоритм выглядит так:

---

**Алгоритм 7.5.** Агломеративная кластеризация Ланса-Уильямса

---

- 1: инициализировать множество кластеров  $C_1$ :  
 $t := 1; C_t = \{\{x_1\}, \dots, \{x_t\}\};$
  - 2: **для всех**  $t = 2, \dots, \ell$  ( $t$  — номер итерации):  
 3:   найти в  $C_{t-1}$  два ближайших кластера:  
 $(U, V) := \arg \min_{U \neq V} R(U, V);$   
 $R_t := R(U, V);$
  - 4:   изъять кластеры  $U$  и  $V$ , добавить слитый кластер  $W = U \cup V$ :  
 $C_t := C_{t-1} \cup \{W\} \setminus \{U, V\};$
  - 5:   **для всех**  $S \in C_t$   
 6:     вычислить расстояние  $R(W, S)$  по формуле Ланса-Уильямса;
- 

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.

Расстояние ближнего соседа:	$\alpha_U = \alpha_V = \frac{1}{2}, \beta = 0, \gamma = -\frac{1}{2};$
$R^b(W, S) = \min_{w \in W, s \in S} \rho(w, s);$	
Расстояние дальнего соседа:	$\alpha_U = \alpha_V = \frac{1}{2}, \beta = 0, \gamma = \frac{1}{2};$
$R^d(W, S) = \max_{w \in W, s \in S} \rho(w, s);$	
Среднее расстояние:	$\alpha_U = \frac{ U }{ W }, \alpha_V = \frac{ V }{ W }, \beta = \gamma = 0;$
$R^c(W, S) = \frac{1}{ W  S } \sum_{w \in W} \sum_{s \in S} \rho(w, s);$	
Расстояние между центрами:	$\alpha_U = \frac{ U }{ W }, \alpha_V = \frac{ V }{ W }, \beta = -\alpha_U \alpha_V, \gamma = 0;$
$R^m(W, S) = \rho^2 \left( \sum_{w \in W} \frac{w}{ W }, \sum_{s \in S} \frac{s}{ S } \right);$	
Расстояние Уорда:	$\alpha_U = \frac{ S + U }{ S + W }, \alpha_V = \frac{ S + V }{ S + W },$ $\beta = \frac{- S }{ S + W }, \gamma = 0.$
$R^w(W, S) = \frac{ S  W }{ S + W } \rho^2 \left( \sum_{w \in W} \frac{w}{ W }, \sum_{s \in S} \frac{s}{ S } \right);$	

...

Возможных вариантов слишком много, и на первый взгляд все они кажутся достаточно разумными. Возникает вопрос: какой из них предпочтительнее? Рассмотрим несколько дополнительных свойств, характеризующих качество кластеризации.

**Свойство монотонности.** Обозначим через  $R_t$  расстояние между ближайшими кластерами, выбранными на  $t$ -м шаге для слияния. Говорят, что функция расстояния  $R$  обладает свойством монотонности, если при каждом слиянии расстояние между объединяемыми кластерами только увеличивается:  $R_2 \leq \dots \leq R_l$

**Теорема 7.1 (Миллиган, 1979).** Если выполняются следующие три условия, то кластеризация является монотонной:

1.  $\alpha_U \geq 0, \alpha_V \geq 0$
2.  $\alpha_U + \alpha_V + \beta \geq 1$
3.  $\min \alpha_U, \alpha_V + \gamma \geq 0.$

Из перечисленных выше расстояний только  $R^w$  не является монотонным.

**Свойство редуктивности.** Самой трудоёмкой операцией в Алгоритме является поиск пары ближайших кластеров на шаге 3. Идея ускорения алгоритма заключается в том, чтобы перебирать лишь наиболее близкие пары. Задаётся параметр  $\delta$ , и перебор ограничивается сокращённым множеством пар  $(U, V) : R(U, V) \leq \delta$ . Когда все такие пары будут исчерпаны, параметр  $\delta$  увеличивается, и формируется новое сокращённое множество пар. И так далее, до полного слияния всех объектов в один кластер. Доказано, что этот алгоритм приводит к той же кластеризации, если расстояние  $R$  обладает свойством редуктивности:

**Оп. 7.1 (Брюиниш, 1978).** Расстояние  $R$  называется *редуктивным*, если для любого  $\delta > 0$  и любых  $\delta$ -близких кластеров  $U$  и  $V$  объединение  $\delta$ -окрестностей  $U$  и  $V$  содержит в себе  $\delta$ -окрестность кластера  $W = U \cup V$ :

$$\{S \mid R(U \cup V, S) < \delta, R(U, V) \leq \delta\} \subseteq \{S \mid R(S, U) < \delta \text{ или } R(S, V) < \delta\}.$$

**Теорема 7.2 (Диде и Моро, 1984).** Если выполняются следующие три условия, то расстояние  $R$  является редуктивным:

- 1)  $\alpha_U \geq 0, \alpha_V \geq 0;$
- 2)  $\alpha_U + \alpha_V + \min\{\beta, 0\} \geq 1;$
- 3)  $\min\{\alpha_U, \alpha_V\} + \gamma \geq 0.$

Из перечисленных выше расстояний только  $R^H$  не является редуктивным.

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

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

**Кластеризация с помощью ЕМ-алгоритма** (с 119)

**Разжевывание для дебиков:**

Задача разделения смеси заключается в том, чтобы, имея выборку  $X_m$  случайных и независимых наблюдений из смеси  $p(x)$ , зная число  $k$  и функцию  $\phi$ , оценить вектор параметров  $\theta = (w_1, \dots, w_k, \theta_1, \dots, \theta_k)$ .

Принцип максимума правдоподобия «в лоб», приводит к слишком громоздкой оптимизационной задаче, поэтому исп. алгоритм ЕМ (expectation-maximization). Идея алгоритма заключается в следующем. Искусственно вводится вспомогательный вектор скрытых (hidden) переменных  $G$ , обладающий двумя **замечательными свойствами**: он может быть вычислен, если известны значения вектора параметров  $\Theta$ , плюс, поиск максимума правдоподобия сильно упрощается, если известны значения скрытых переменных.

ЕМ-алгоритм состоит из итерационного повторения двух шагов. На Е-шаге вычисляется ожидаемое значение (expectation) вектора скрытых переменных  $G$  по текущему приближению вектора параметров  $\theta$ . На М-шаге решается задача максимизации правдоподобия (maximization) и находится следующее приближение вектора  $\theta$  по текущим значениям векторов  $G$  и  $\theta$ .

---

**Алгоритм 2.1.** Общая идея ЕМ-алгоритма

- 1: Вычислить начальное приближение вектора параметров  $\Theta$ ;
  - 2: **повторять**
  - 3:  $G := EStep(\Theta);$
  - 4:  $\Theta := MStep(\Theta, G);$
  - 5: **пока**  $\Theta$  и  $G$  не стабилизируются.
- 

Как удачно, что задача кластеризации сводится к разделению смеси распределений по конечной выборке.

**Гипотеза 7.1 (о вероятностной природе данных).** Объекты выборки  $X^\ell$  появляются случайно и независимо согласно вероятностному распределению, представляющему собой смесь распределений

$$p(x) = \sum_{y \in Y} w_y p_y(x), \quad \sum_{y \in Y} w_y = 1,$$

где  $p_y(x)$  — функция плотности распределения кластера  $y$ ,  $w_y$  — неизвестная априорная вероятность появления объектов из кластера  $y$ .

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

Преимущество эллиптических гауссианов в том, что они обходят проблему выбора нормировки признаков. Нормировать можно немного по-разному, причём результат кластеризации существенно зависит от нормировки. Пока не произведена кластеризация, трудно понять, какая нормировка лучше. При использовании эллиптических гауссианов оптимальная нормировка подбирается самим алгоритмом кластеризации, индивидуально для каждого кластера.

При этих предположениях задача кластеризации совпадает с задачей разделения смеси вероятностных распределений (алгоритм ниже) Напомним, что ЕМ-алгоритм заключается в итерационном повторении двух шагов.

**На Е-шаге** по формуле Байеса вычисляются скрытые переменные  $g_{iy}$ . Значение  $g_{iy}$  равно апостериорной вероятности того, что объект  $x_i \in X_l$  принадлежит кластеру  $y \in Y$ .

**На М-шаге** уточняются параметры каждого кластера  $(\mu_y, \Sigma_y)$ , при этом существенно используются скрытые переменные  $g_{iy}$ . Для простоты предполагается, что число кластеров известно заранее.

---

**Алгоритм 7.3.** Кластеризация с помощью ЕМ-алгоритма

---

- 1: начальное приближение для всех кластеров  $y \in Y$ :  
 $w_y := 1/|Y|$ ;  
 $\mu_y :=$  случайный объект выборки;  
 $\sigma_{yj}^2 := \frac{1}{\ell|Y|} \sum_{i=1}^{\ell} (f_j(x_i) - \mu_{yj})^2$ ,  $j = 1, \dots, n$ ;
  - 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;
- 

Общий вид ЕМ-алгоритма (когда распределение берется из произвольного параметрического семейства):

---

35

---

---

**Алгоритм 2.2.** ЕМ-алгоритм с фиксированным числом компонент

---

**Вход:**

выборка  $X^m = \{x_1, \dots, x_m\}$ ;  
 $k$  — число компонент смеси;  
 $\Theta = (w_j, \theta_j)_{j=1}^k$  — начальное приближение параметров смеси;  
 $\delta$  — параметр критерия останова;

**Выход:**

$\Theta = (w_j, \theta_j)_{j=1}^k$  — оптимизированный вектор параметров смеси;

---

- 1: **ПРОЦЕДУРА**  $\text{EM}(X^m, k, \Theta, \delta)$ ;
  - 2: **повторять**
  - 3: Е-шаг (expectation):  
для всех  $i = 1, \dots, m$ ,  $j = 1, \dots, k$   
$$g_{ij}^0 := g_{ij}; \quad g_{ij} := \frac{w_j \varphi(x_i; \theta_j)}{\sum_{s=1}^k w_s \varphi(x_i; \theta_s)};$$
  - 4: М-шаг (maximization):  
для всех  $j = 1, \dots, k$   
$$\theta_j := \arg \max_{\theta} \sum_{i=1}^m g_{ij} \ln \varphi(x_i; \theta); \quad w_j := \frac{1}{m} \sum_{i=1}^m g_{ij};$$
  - 5: **пока**  $\max_{i,j} |g_{ij} - g_{ij}^0| > \delta$ ;
  - 6: **вернуть**  $(w_j, \theta_j)_{j=1}^k$ ;
- 

**Формула Ланса-Уильямса** (с 122)

Формула Ланса-Уильямса нужна для быстрого пересчета межклusterных 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.

## 17.2 Алгоритмы k-Means и DBSCAN.

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

Теория:

[k-Means](#) (с 120)

**Метод k-средних**, представленный ниже, является упрощением ЕМ-алгоритма. Главное отличие в том, что в ЕМ-алгоритме каждый объект  $x_i$  распределяется по всем кластерам с вероятностями  $g_{iy} = P\{y_i = y\}$ . В алгоритме k-средних (k-means) каждый объект жёстко приписывается только к одному кластеру. Второе отличие в том, что в k-means форма кластеров не настраивается. Возможен и обобщённый вариант k-means, в котором будут определяться размеры кластеров вдоль координатных осей.

Таким образом, ЕМ и k-means довольно плавно «перетекают» друг в друга, позволяя строить различные «промежуточные» варианты этих двух алгоритмов.

Алгоритм k-means крайне чувствителен к выбору начальных приближений центров. Для формирования начального приближения можно выделить к наиболее удалённых точек выборки: первые две точки выделяются по максимуму всех попарных расстояний; каждая следующая точка выбирается так, чтобы расстояние от неё до ближайшей уже выделенной было максимально.

Другая рекомендация – выполнить кластеризацию несколько раз, из различных случайных начальных приближений и выбрать кластеризацию с наилучшим значением заданного функционала качества.

#### Алгоритм 7.4. Кластеризация с помощью алгоритма k-средних

1: сформировать начальное приближение центров всех кластеров  $y \in Y$ :

$\mu_y$  — 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.

Минусы:

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  $\text{minPts}$  соседей, включая ее саму, на расстоянии не больше  $\varepsilon$ . Назовем этих соседей прямо-достижимыми из точки  $p$ . По определению все точки могут быть прямо-достижимы 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  $q$ ).  
n

- 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:  $\text{minPts}$  и  $\varepsilon$

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  $\varepsilon$  и метрику. Особенно в пространствах большой размерности.
3. Не умеет в кластеры разной плотности. Так как  $\text{minPts}$  и  $\varepsilon$  одинаковые для всех кластеров.

## 18 Теоретический минимум

### 18.1 В чем 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 напишем вкратце что это за задачи и пример.

1. 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.
2. 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.
3. 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 программ.
4. 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.

### 18.2 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 признаконое 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 признаконым 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 искомому 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).

### 18.3 Запишите формулы для линейной модели регрессии и для среднеквадратичной ошибки.

$x_i$  — вектор признаков  $i$ -го объекта,  $y_i$  — значение целевой функции для него,  $\langle w, x \rangle$  — скалярное произведение векторов  $w$  и  $x$ .  $X$  - матрица из  $x_i$ ,  $Y$  - вектор из  $y_i$ .

Формула, по которой считаем прогноз целевой функции:

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

Среднеквадратичная ошибка:

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

Решение задачи, если просто хотим минимизировать квадратичную норму ошибки:

$$w = (X^T X)^{-1} X^T Y.$$

### 18.4 Что такое градиент? Какое его свойство используется при минимизации функций?

Градиент - вектор, своим направлением указывающий направление наибольшего возрастания некоторой величины  $\phi$ , значение которой меняется от одной точки пространства к другой (скалярного поля), а по величине (модулю) равный скорости роста этой величины в этом направлении.

С математической точки зрения на градиент можно смотреть как на:

1. Коэффициент линейности изменения значения функции многих переменных от изменения значения аргумента
2. Вектор в пространстве области определения скалярной функции многих переменных, составленный из частных производных
3. Строки Матрицы Якоби содержат градиенты составных скалярных функций из которых состоит векторная функция многих переменных

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

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

Свойство для минимизации:  $-\nabla f(x)$  — направление наибольшего убывания функции.

### 18.5 Запишите формулу для одного шага градиентного спуска. Как модифицировать градиентный спуск для очень большой выборки?

Пусть  $x_i$  — признаковое описание  $i$ -го объекта,  $y_i$  — значение целевой функции для него,  $f(w, x_i)$  — функция предсказания целевой функции, зависящая еще от неких настраиваемых весов,  $L(y, \hat{y})$  — функция потерь для одного объекта. Тогда формула для шага градиентного спуска выглядит так

$$w_t = w_{t-1} - \beta_t \sum_{i=1}^n \nabla_w L(y_i, f(w_{t-1}, x_i)),$$

где  $\beta_t$  выбирается неким образом на каждом шаге. Например, может быть просто небольшой константой.

Для большой выборки можно на каждом шаге суммировать не по всем объектам, а по некоторому небольшому случайному подмножеству элементов (может быть даже состоящему только из одного элемента).

### 18.6 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.

## 18.7 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:

1. Параметр регуляризации  $\lambda$  (при использовании регуляризатора)
2. Степень полинома в задаче регрессии с семейством алгоритмов, заданным множеством полиномов определенной степени

Параметры – матрица весов и вектор смещений. В решающих деревьях гиперпараметры: максимальная глубина, минимальное число элементов в листьях (и вообще выбор условия в листе), а параметрами служат элементы разбиений: признаки и пороги.

## 18.8 Что такое регуляризация? Чем на практике отличается L1-регуляризация от L2?

Регуляризация – метод борьбы с переобучением. Она накладывает штраф за сложность модели, выражаящийся в дополнительных слагаемых в функции потерь умноженных на коэффи. регуляризации. L1 регуляризация обнуляет некоторые веса, позволяя таким образом отбирать признаки в то время, как L2 просто ограничивает значения весов.

## 18.9 Запишите формулу для линейной модели классификации. Что такое отступ?

Линейная модель классификации определяется следующим образом:

$$a(x) = \text{sign}(\langle w, x \rangle + w_0) = \text{sign}\left(\sum_{j=1}^d w_j x_j + w_0\right)$$

В случае бинарной классификации удобнее всего в качестве функционала ошибки использовать долю правильных ответов. Т.е.

$$Q(a, X) = \frac{1}{l} \sum_{i=1}^l [a(x_i) = y_i].$$

Нам удобнее решать задачу минимизации, поэтому будем вместо этого использовать долю неправильных ответов:

$$Q(a, X) = \frac{1}{l} \sum_{i=1}^l [a(x_i) \neq y_i] = \frac{1}{l} \sum_{i=1}^l [\text{sign}\langle w, x_i \rangle \neq y_i] = \frac{1}{l} \sum_{i=1}^l [y_i \langle w, x_i \rangle < 0] \rightarrow \min_w$$

$M_i = y_i \langle w, x_i \rangle$  – называется отступом. Знак отступа говорит о корректности ответа классификатора, а его абсолютная величина характеризует степень уверенности классификатора в своём ответе.

## 18.10 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.

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

$$\text{Recall} = \frac{TP}{TP+FN} \text{ -- доля сбитых самолетов.}$$

## 18.11 Что такое ROC-AUC? Как построить ROC-кривую?

ROC-кривая - это кривая, показывающая зависимость доли верных положительных ответов от доли ложных положительных ответов при варьировании порога решающего правила.

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.

**18.12 Запишите функционал логистической регрессии. Как он связан с методом максимума правдоподобия?**

Пусть  $y_i^j = I(\text{класс } i\text{-го объекта } - j)$ , а  $\hat{y}_i^j$  - наш ответ для вероятности принадлежности  $i$ -го объекта  $j$ -му классу.  $n$  - число объектов,  $k$  - число классов. Тогда функция потерь

$$L = - \sum_{i=1}^n \sum_{j=1}^k y_i^j \ln \hat{y}_i^j.$$

Пропотенцируем эту функцию потерь, получим правдоподобие

$$\exp(-L) = \prod_{i=1}^n \prod_{j=1}^k \left( \hat{y}_i^j \right)^{y_i^j}.$$

$$p(y^1, y^2, \dots, y^k) = \prod_{j=1}^k \left( \hat{y}_i^j \right)^{y^j} \text{ --- плотность мультиномиального распределения с параметрами } \hat{y}_i^j.$$

Более простой и привычный вариант с  $k = 2$ :

$$L = - \sum_{i=1}^n y_i \ln \hat{y}_i + (1 - y_i) \ln(1 - \hat{y}_i),$$

$$\exp(-L) = \prod_{i=1}^n (\hat{y}_i)^{y_i} (1 - \hat{y}_i)^{1-y_i},$$

$$p(y) = (\hat{y}_i)^y (1 - \hat{y}_i)^{1-y} \text{ --- плотность распределения бернуlli с параметром } \hat{y}_i.$$

**18.13 Запишите задачу метода опорных векторов для линейно неразделимого случая. Как функционал этой задачи связан с отступом классификатора?**

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

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

Найдите теорминимум. Впрочем ничего нового.

**18.14 Опишите жадный алгоритм обучения решающего дерева.**

Дерево строится рекурсивно, в каждом узле выбирается оптимальный признак и оптимальное разбиение по его значениям в соответствии с выбранным критерием (энтропийный и gini в классификации, MSE в регрессии).

**18.15 Почему с помощью решающего дерева можно достичь нулевой ошибки на обучающей выборке без повторяющихся объектов?**

Поскольку объекты не повторяются, каждому набору признаков из представленных в выборке можно сопоставить ровно одно значение целевой переменной, тогда в ветвях дерева можно перебрать все наборы признаков и дать соответствующие им ответы в листьях.

## 18.16 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:  $\frac{1}{n} \sum_{i=1}^n a_{X_i}(x)$ .

## 18.17 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.

## 18.18 Как в градиентном бустинге обучаются базовые алгоритмы? Что такое сокращение шага?

Алгоритмы обучаются последовательно, каждый следующий обучается на антиградиент ошибки предыдущего. Построенный таким образом алгоритм отвечает суммой взвешенных ответов базовых алгоритмов (или, что эквивалентно, берётся не целый антиградиент, а взвешенный). Сокращение шага - уменьшение веса нового базового алгоритма на каждом шаге.

## 18.19 Зачем нужен backprop, что такое производная вектора по вектору?

Backprop - это обратное распространение ошибки. Нужно для вычисления производных по выходу в нейронах скрытых слоёв. Производная вектора по вектору - матрица Якоби  $\frac{\partial f(\vec{x})}{\partial \vec{x}} = \begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & & \vdots \\ \frac{\partial f_m}{\partial x_1} & \cdots & \frac{\partial f_m}{\partial x_n} \end{pmatrix}$

## 18.20 Чем нейросеть отличается от линейной модели, приведите примеры нейросетей?

Нейросеть содержит скрытые слои, в то время как линейная модель представляет собой нечто вроде однослойной нейросети. Более того, способности современных нейросетей далеко не ограничиваются линейным преобразованием. Используется большой набор других слоёв: нелинейности, нормализации, регуляризации, dropout/dropconnect, свёртки, pooling, рекуррентные слои.

## 18.21 Объясните идею weight sharing на примере сверточного слоя, почему эта техника работает именно с изображениями? Какое свойство входных объектов мы учитываем?

**Объяснение с анимацией** В пределах одного сверточного слоя применяются одни и те же веса свёртки: к каждому окну и к каждому каналу входных данных. На изображениях это соответствует тому, что свёртка находит контуры каких-то деталей изображения, независимо от их местоположения и цвета. Также важно, что цветовые каналы согласованы между собой по контурам, образуя единое целое.

## 18.22 В чём 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.

## 18.23 Как работает метод K-Means?

Это метод решающий 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.

## 18.24 Как работает метод t-SNE?

Эмбединг в пространство меньшей размерности с сохранением свойства локальности. Есть параметр перплексия, которая условно задает ожидаемое количество соседей.

Если хочется построить, то можно сказать следующее. Мы задаем меру близости точек в многомерном пространстве и меру близости точек в пространстве меньшей размерности. А дальше подгоняем образы точек выборки так, чтобы эти меры были как можно ближе.

**Алгоритм t-SNE** в упрощенном виде можно представить следующим псевдокодом:

Data: набор данных  $X = x_1, x_2, \dots, x_n$ ,

параметр функции потерь: перплексия  $P_{\text{perp}}$ ,

Параметры оптимизации: количество итераций  $T$ , скорость обучения  $\eta$ , момент  $\alpha(t)$ .

Result: представление данных  $Y(T) = y_1, y_2, \dots, y_n$  (в 2D или 3D).

1. вычислить попарное сходство  $p_{j|i}$  с перплексией  $P_{\text{perp}}$  (используя формулу 1)
2. установить  $p_{ij} = \frac{(p_{j|i} + p_{i|j})}{2n}$ .
3. инициализировать  $Y(0) = \{y_1, y_2, \dots, y_n\}$  точками нормального распределения (mean=0, sd=1e-4)
4. for  $t = 1$  to  $T$  do
  - (a) вычислить сходство точек в пространстве отображения  $q_{ij}$  (по формуле 4)
  - (b) вычислить градиент  $\frac{\partial \text{Cost}}{\partial y}$  (по формуле 5)
  - (c) установить  $Y(t) = Y(t-1) + \tau \frac{\partial \text{Cost}}{\partial y} + \alpha(t)(Y(t-1) - Y(t-2))$ .

## 18.25 Запишите постановку задачи в методе главных компонент.

Пусть  $n$  — количество объектов,  $k$  — число признаков для каждого объекта,  $X$  — матрица признаков  $n \times k$ . Хотим уменьшить число признаков до  $m < k$ , построив некое линейное отображение из  $\mathbb{R}^k$  в  $\mathbb{R}^m$ . При этом требуется сделать так, чтобы можно было с минимальными потерями восстановить старые признаки. Пусть  $Z$  — матрица новых признаков ( $n \times m$ ), а  $U$  — матрица восстановления ( $m \times k$ ). Тогда задача ставится так

$$\|ZU - X\|^2 \rightarrow \min_{Z, U},$$

где под нормой подразумевается норма Фробениуса (корень из суммы квадратов элементов).