

**Машинное обучение**  
**5 семестр**  
**Лектор: Нейчев Р.Г.**  
**осень 2022**

**Авторы билетов (лучшие котики):**

Спицын Николай  
Савичев Дмитрий  
Подзорова Полина  
Чубенко Полина  
Климанова Ирина  
Клячин Артемий  
Сбродов Егор

Special thanks to авторам теормина прошлого года, Даниил Гагаринов  
и Артур Кулапин

# Содержание

|      |   |    |
|------|---|----|
| 0.1  | Постановка задач обучения с учителем (supervised learning) . . . . .  | 5  |
| 0.2  | Задачи обучения без учителя. Назвать хотя бы две. . . . .   | 5  |
| 0.3  | Что означает свойство i.i.d.? . . . . .   | 6  |
| 0.4  | Основная идея наивного Байесовского классификатора. В чём его наивность? . . . . .  | 6  |
| 0.5  | Запишите формулы для модели линейной регрессии и для среднеквадратичной ошибки. . . . .   | 6  |
| 0.6  | Запишите формулу для одного шага градиентного спуска. Как модифицировать градиентный спуск для очень большой выборки? . . . . .                           | 6  |
| 0.7  | Что такое правдоподобие, метод максимального правдоподобия? Является ли правдоподобие вероятностью? . . . . .   | 7  |
| 0.8  | Что такое кросс-валидация? На что влияет количество блоков в кросс-валидации? . . . . .   | 7  |
| 0.9  | Что такое переобучение и недообучение? Как их можно детектировать? . . . . .  | 8  |
| 0.10 | Чем гиперпараметры отличаются от параметров? Что является параметрами и гиперпараметрами в линейных моделях и в решающих деревьях? . . . . .              | 8  |
| 0.11 | Что такое регуляризация? Чем на практике отличается $L_1$ -регуляризация от $L_2$ ? . . . . .   | 9  |
| 0.12 | Учитывается ли коэффициент сдвига $\omega$ в регуляризаторе? Почему? . . . . .  | 9  |
| 0.13 | Почему линейные модели рекомендуется применять к выборке с нормированными значениями признаков? . . . . .   | 9  |
| 0.14 | Запишите формулу для линейной модели классификации. Что такое отступ? . . . . .   | 9  |
| 0.15 | Что такое точность и полнота? Почему нужно учитывать их вместе? . . . . .   | 10 |
| 0.16 | В задаче бинарной классификации доля одного класса составляют 95% выборки. Какие метрики разумно использовать для оценки работы модели? Почему? . . . . . | 11 |
| 0.17 | Что такое ROC-AUC? Как построить ROC-кривую? . . . . .  | 11 |
| 0.18 | Запишите функционал логистической регрессии. Как он связан с методом максимума правдоподобия? . . . . .   | 11 |
| 0.19 | Идея метода опорных векторов (в случае разделимой выборки). . . . .   | 12 |
| 0.20 | Опишите жадный алгоритм обучения решающего дерева. . . . .  | 12 |
| 0.21 | Почему с помощью решающего дерева можно достичь нулевой ошибки на обучающей выборке без повторяющихся объектов? . . . . .                                 | 13 |
| 0.22 | Если в лист дерева попали объекты разных классов, то какие предсказания нужно выдавать в этом листе? Почему? . . . . .                                    | 13 |
| 0.23 | Какое предсказание нужно выдавать в листе дерева в задаче регрессии если мы минимизируем MSE? а в случае MAE? . . . . .                                   | 13 |
| 0.24 | Что такое bagging? . . . . .  | 13 |
| 0.25 | Что такое случайный лес? Чем он отличается от бэггинга над решающими деревьями? . . . . .   | 13 |
| 0.26 | Как в градиентном бустинге обучаются базовые алгоритмы? . . . . .   | 14 |
| 0.27 | Зачем нужен backprop, что такое производная вектора по вектору? . . . . .   | 14 |
| 0.28 | Опишите принцип работы свёрточного слоя (CNN). . . . .  | 14 |
| 0.29 | В чём недостатки полно связанных нейронных сетей какая мотивация к использованию свёрточных? . . . . .  | 15 |
| 0.30 | Опишите принцип работы базового рекуррентного слоя (RNN). . . . .   | 15 |
| 0.31 | Что такое dropout? . . . . .  | 16 |
| 0.32 | Как dropout и batch normalization меняют свое поведение при эксплуатации модели (в режиме inference)? . . . . .   | 16 |
| 0.33 | Запишите постановку задачи в методе главных компонент. . . . .  | 16 |
| 1    | Machine Learning problem statement. Regression, Classification, examples.   | 17 |
| 2    | How to measure quality in classification: accuracy, balanced accuracy, precision, recall, f1-score, ROC-AUC, multiclass extensions                        | 18 |

|   |    |
|---|----|
| 3 How to measure quality in regression: MSE, MAE, R2.   | 22 |
| 4 Maximum likelihood estimation, how is it related to regression and classification   | 23 |
| 5 Naive bayesian classifier, how does it work   | 23 |
| 6 K-nearest neighbours classifier, how does it work   | 24 |
| 7 Linear regression. Problem statement for the MSE loss function case. Analytical solution. Gauss-Markov theorem. Gradient approach in linear regression. | 26 |
| 8 Regularization in linear models: L1 и L2, their properties. Probabilistic interpretation.   | 27 |
| 9 Logistic regression. Equivalence of MLE approach and logistic loss minimization.  | 28 |
| 10 Multiclass classification. One-vs-one, one-vs-all, their properties.   | 29 |
| 11 Support vector machine. Optimization problem for SVM. Kernel trick. Kernel properties.   | 31 |
| 12 Principal component analysis. Relations to SVD. Eckart-Young theorem. How to apply PCA in practice.  | 36 |
| 13 Train, validation and test stages of model development. Overfitting problem, ways to detect it.  | 39 |
| 14 Validation strategies. Cross validation. Data leaks.   | 41 |
| 15 Bias-variance tradeoff.  | 42 |
| 16 Decision tree construction procedure.  | 44 |
| 17 Information criteria. Entropy criteria, Gini impurity.   | 47 |
| 18 Ensembling methods. Bootstrap. Bagging.  | 48 |
| 19 Random Forest, Random subspace method.   | 49 |
| 20 Boosting and gradient boosting. Main idea, gradient derivation.  | 50 |
| 21 Matrix calculus and matrix derivatives. How to get the derivative of matrix/dot product  | 52 |
| 22 Backpropagation, chain rule.   | 53 |
| 23 Neural network concept. Fully-Connected layer (FC). Logistic regression as simple NN. XOR problem.   | 55 |
| 24 Losses for NNs: logistic loss, cross-entropy.  | 56 |
| 25 Activation functions, their impact on the network, computational complexity. Softmax and LogSoftmax activations, numerical stability.                  | 57 |
| 26 Optimization methods in Deep Learning. Gradient descent, SGD, its upgrades: Momentum, RMSProp, Adam.   | 58 |

|  |           |
|--|-----------|
| <b>27 Regularization in Deep Learning: Dropout, Batch<br/>Normalization. Differences in training and evaluation stages.</b>  | <b>59</b> |
| <b>28 Vanilla Recursive NN cell. Backpropagation through RNN. Vanishing gradient problem.<br/>Potential solutions.</b>   | <b>61</b> |
| <b>29 LSTM/GRU, memory concept, gates ideas.</b>   | <b>63</b> |
| <b>30 Matrix convolution. Convolutional layer, backpropagation<br/>through it. Hyperparameters of Convs. 1x1 convolutions, comparison to FC layers.<br/>Max/Average Pooling.</b> | <b>64</b> |
| <b>31 Main ideas of AlexNet, VGG, Inception (GoogLeNet), ResNet architectures.</b>   | <b>65</b> |
| <b>32 Geometrical methods in ML. Clustering problem. IsoMap, LLE, DBSCAN, k-means,<br/>t-SNE</b>   | <b>66</b> |

## 0.1 Постановка задач обучения с учителем (supervised learning).

Постановка задачи с «учителем» подразумевает наличие целевых меток для предсказания (targets).

Определим следующее:

Training set  $\mathcal{L} = \{x_i, y_i\}_{i=1}^n$ , где

- ▷  $(x \in \mathbb{R}^p, y \in \mathbb{R})$  для регрессии
- ▷  $x_i \in \mathbb{R}^p, y_i \in \{0, 1\}$  для бинарной классификации
- ▷  $x_i \in \mathbb{R}^p, y_i \in \{c_1, \dots, c_n\}$  для многоклассовой классификации

Модель  $f(X)$ , которая предсказывает какое-то значение для каждого объекта.

Функцию потерь  $Q(X, y, f)$ , которую мы будем минимизировать.

Формальнее, пусть имеется семейство моделей  $\mathcal{F}$ . Допустим, что оно параметризовано (может и без параметризации) вектором  $\theta \in \Theta$ , тогда задача оптимизации: найти  $f \in \mathcal{F}$ , что  $f = \arg \min_{\theta} Q(X, y, f_{\theta})$ . Как пример, линейные регрессионные модели параметризуются векторами весов,  $\Theta = \mathbb{R}^{p+1}$  (для bias term).

Примеры задач обучения с учителем:

- ▷ 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.
- ▷ 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.

## 0.2 Задачи обучения без учителя. Назвать хотя бы две.

Задача обучения без учителя: у нас нет таргета.

Примеры таких задач:

- ▷ 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 одежду. Ясно, что парням и девушкам нужно показывать разные рекомендации, поэтому множество потенциальных покупателей было бы неплохо разбить на кластеры.
- ▷ Уменьшение размерности — задача обучения без учителя, в которой хочется построить отображение из многомерного пространства в пространство существенно меньшей размерности с минимальными “потерями”. Хотим либо уметь 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.

### 0.3 Что означает свойство i.i.d.?

i.i.d = независимые и одинаково распределенные

Объект —  $p$ -мерный вектор, порожденный из некоторого распределения (то есть случайной величиной), тут подразумевается независимость объектов как независимость таких случайных векторов.

Однаково распределенные: порождены одинаковым процессом (с одинаковым распределением)

Это условие мы требуем от объектов

### 0.4 Основная идея наивного Байесовского классификатора. В чём его наивность?

Основная идея: мы хотим использовать теорему Байеса. Для этого нам надо чтобы признаки были независимы (а это обычно неправда, например, температура в Цельсиях и Фаренгейтах).

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

Пусть у нас есть матрица объектов  $X = (x_0, x_1, \dots, x_n)^T$  и столбец таргетов  $Y = (y_1, \dots, y_n)^T$

Тогда модель линейной регрессии записывается так:

$$f_w(X) = Xw = \hat{Y} \approx Y$$

Среднеквадратичная ошибка записывается так:  $Q_{MSE}(X) = \frac{1}{n}(Y - Xw)^T(Y - Xw)$ , где  $n$  — мощность выборки.

Оптимальная оценка для просто квадратичной нормы ищется так:

$$\hat{\theta} = (Z^T Z)^{-1} Z^T X.$$

Рассмотрим

$$\Delta X - Z\theta^2 = (X - Z\theta)^T(X - Z\theta) = X^T X - 2X^T Z\theta + \theta^T Z^T Z\theta$$

Заметим, что перед нами «квадратный трехчлен» по  $\theta$ . Продифференцируем по  $\theta$ :

$$-2(X^T Z)_i + 2(\theta^T Z^T Z)_i = 0 \implies \theta^T Z^T Z - X^T Z = \vec{0} \implies Z^T Z\theta = Z^T X \implies (Z^T Z)^{-1} Z^T X$$

— несмешённая, то есть  $\mathbb{E} = \theta$ ; и  $= \sigma^2 (Z^T Z)^{-1}$ .

Теорема о наилучшей оценке в классе линейных оценок (б/д).

Наилучшая или «оптимальная» — достигается равенство в неравенстве Рао-Крамера (или обладает минимальной дисперсией).

Пусть  $t = T\theta$ , где  $T \in \text{Mat}_{m \times k}(\mathbb{R})$ . Тогда ОНК имеет вид  $\hat{t} = T$  и является оптимальной оценкой  $t$  в классе линейных несмешённых оценок, то есть представимых в виде  $B \cdot X$ .

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

Градиент — вектор, направление которого совпадает с направлением наибольшего возрастания величины  $\varphi$ , значение которой меняется от одной точки пространства к другой (скалярного поля), а по величине (модулю) равный скорости роста этой величины в этом направлении.

$\nabla f(x)$  — направление наибольшего убывания функции.

Введем следующие переменные:

▷  $n$  — число объектов в выборке

- ▷  $x_i$  — вектор признаков  $i$ -го объекта
- ▷  $y_i$  — таргет
- ▷  $w$  — вектор весов (его ищем для поиска оптимальной модели)
- ▷  $f(w, x)$  — функция предсказания целевой функции
- ▷  $L(y, \hat{y})$  — функция потерь для одного объекта

Тогда формула для градиентного спуска выглядит так: ( $\beta_t$  — шаг градиентного спуска)

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

Шаг градиентного спуска может выбираться на каждом шаге, а может быть константой

Для большой выборки можно на каждом шаге суммировать не по всем объектам, а по некоторому небольшому случайному подмножеству элементов (SGD) (не нужно хранить в памяти значения вектора градиента по всей выборке).

## 0.7 Что такое правдоподобие, метод максимального правдоподобия? Является ли правдоподобие вероятностью?

Правдоподобие  $L(\theta | X, Y) = P(X, Y | \theta)$ , где  $\theta$  — вектор параметров,  $X$  — матрица признаков, а  $Y$  — вектор target.

Таким образом, правдоподобие — это **вероятность** получить такую матрицу признаки-цели при данном значении вектора параметров распределения объектов.

Метод максимального правдоподобия — метод, при котором правдоподобие стараются максимизировать (спасибо, кэп)

## 0.8 Что такое кросс-валидация? На что влияет количество блоков в кросс-валидации?

Для работы модели нужно определить ее гиперпараметры.

Первая идея (не кросс-валидация): обучить модель по части train и по предсказаниям на второй части определить оптимальные гиперпараметры. Минус такого подхода в том, что модель будет зависеть от случайного выбора куска трейна (а вдруг объекты в train упорядочены по алфавиту и не все объекты попадут в обучающую выборку при определении гиперпараметров. Модель с этими гиперпараметрами может показать плохой результат на всей обучающей выборке).

Вторая идея (кросс-валидация): часть объектов из train идет на валидацию, часть на train, затем данный процесс повторяется несколько раз (часть с валидацией меняется). Тогда мы можем измерить дисперсию ошибки для каждого набора гиперпараметров и сам лосс, чтобы выбрать оптимальный набор.

Далее речь идет о K-fold кросс-валидации, где  $K$  — число запусков кросс-валидации и одновременно то, на сколько примерно равных кусков мы бьем train. Тогда в рамках каждого запуска один из  $K$  блоков является валидацией, а остальные — train.

Влияние числа блоков:

- ▷ При увеличении числа блоков дисперсия ответа уменьшается (закономерно, ибо объем данных увеличивается)
- ▷ Аналогичная ситуация с bias — матожидание разности между истинным ответом и выданным алгоритмом — он уменьшается
- ▷ Чем меньше количество блоков, тем быстрее это работает, но тем меньше шанс, что модель подстроится под какой-то кусок данных.

## 0.9 Что такое переобучение и недообучение? Как их можно детектировать?

Недообучение — ситуация, когда модель уловила не все общие закономерности и не способна достаточно точно воспроизвести распределение, из которого создаются объекты.

Переобучение — ситуация, когда модель не только успешно смоделировала распределение, но и включила в него шумовые факторы (то есть переобучилась под выбросы).

Как детектировать это все?

Вспомним про `train` и `test`, будем измерять лосс на них в зависимости от сложности модели (например, число эпох или норма вектора весов в линейной регрессии).

- ▷ Если и на `train`, и на `test` лосс падает, то мы уловили не все основополагающие зависимости  
⇒ недообучение.
- ▷ Если на `train` лосс падает, а на `test` лосс растет, то мы уловили шумовые зависимости, чуждые распределению ⇒ переобучение.

## 0.10 Чем гиперпараметры отличаются от параметров? Что является параметрами и гиперпараметрами в линейных моделях и в решающих деревьях?

Параметры настраиваются непосредственно при обучении (например веса в линейной регрессии), в то время как гиперпараметры фиксированные и изменяются вручную, если мы понимаем, что модель учится плохо.

Линейные модели:

- ▷ Гиперпараметры:
  1. Тип регуляризации (может быть структурным параметром)
  2. Параметр регуляризации  $\lambda$  (при использовании регуляризатора)
  3. Степень полинома в задаче регрессии с семейством алгоритмов, заданным множеством полиномов определенной степени.
- ▷ Параметры:
  1. Матрица весов
  2. Вектор смещений

Решающие деревья:

- ▷ Гиперпараметры:
  1. Максимальная глубина
  2. Минимальное число элементов в листьях (и выбор условия в листе)
- ▷ Параметры:
  1. Признаки (номер признака по которому производится разбиение  $i$ )
  2. Пороги (трешхолд  $t$ )

## 0.11 Что такое регуляризация? Чем на практике отличается $L_1$ -регуляризация от $L_2$ ?

Регуляризация — внесение в Loss члена, цель которого штрафовать за сложность модели. Например, для линейной регрессии такой член имеет вид  $\lambda \cdot g(\omega)$ , где  $g$  — монотонно возрастающая по каждому аргументу функция.

$L_1$ -регуляризация имеет вид  $\lambda \cdot \sum_{i=1}^p |\omega_i| = \lambda \triangleleft \omega_{L_1}$  ( $\lambda > 0$ )

$L_2$ -регуляризация имеет вид  $\lambda \cdot \sqrt{\sum_{i=1}^p |\omega_i|^2} = \lambda \triangleleft \omega_{L_2}$  ( $\lambda > 0$ )

Заметим, что  $L_1$ , что  $L_2$  регуляризации в случае если  $\lambda$  велико стремятся занулить вектор весов, так как иначе вклад члена регуляризации слишком большой. Но  $L_1$  делает это значительно агрессивнее. Таким образом,  $L_1$  стремится отобрать информативные признаки.

Но при этом loss для  $L_1$  регуляризации недифференцируем, в отличие от  $L_2$ , да еще и  $L_2$  имеет аналитическое решение.

## 0.12 Учитывается ли коэффициент сдвига $\omega$ в регуляризаторе? Почему?

Ответ: нет.

**Объяснение:** допустим наша выборка является прямой  $y = kx + 10^6$ , тогда включение bias term в регуляризацию будет стремиться подавить его значение, что не позволит нам добрать нужное его значение. Более того, итоговая модель будет выглядеть как  $Y = WX + b$ , где  $b$  как-то от  $X$  не очень-то зависит, то есть  $W$  отвечает за угадывание направления главного тренда, а  $b$  за его смещение, которое является вторичным.

## 0.13 Почему линейные модели рекомендуется применять к выборке с нормированными значениями признаков?

Разберем два случай, есть ли регуляризация, или ее нет:

- Пусть есть регуляризация, тогда на ненормированной выборке можно получить проблему того, что некоторые веса будут огромными, так как масштабы признаков разные. Это приведет к взрыву нормы вектора весов и к стремлению его подавить (хотя этот признак с огромным коэффициентом мог быть самым информативным). Это большая проблема.
- Пусть нет регуляризации, тогда заметим, что нормализация является линейным отображением, а значит линейная модель может включить ее в себя. Но в любом случае масштаб данных разный, значит веса разные, а значит мы не можем проверить, какой признак для нас значим, так как чем больше по модулю вес, тем он значимее.

Как пример, можно поразмышлять над тем, как связаны веса и корреляции признака и target, в нормированных данных это даст почти прямую взаимосвязь, чем выше корреляция, тем выше значимость (информативность) признака (банально можно вспомнить PCA/SVD).

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

Линейная модель классификации (предсказание класса для одного объекта  $x$ ):

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

Далее, чтобы было проще, добавим к матрице признаков единичный столбец, который будет отвечать за сдвиг  $w_0$ .

В случае бинарной классификации удобнее всего в качестве функционала ошибки использовать долю правильных ответов:

$$Q(a, X, y) = \frac{1}{l} \sum_{i=1}^l [a(x_i) = y_i] \quad (X — \text{матрица, состоящая из } x_i)$$

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

$$\begin{aligned} Q(a, X, y) &= \frac{1}{l} \sum_{i=1}^l [a(x_i) \neq y_i] = \frac{1}{l} \sum_{i=1}^l [\text{sign}(\langle w, x_i \rangle) \neq y_i] = \\ &= \frac{1}{l} \sum_{i=1}^l [y_i \langle w, x_i \rangle < 0] \longrightarrow \min_w. \end{aligned}$$

Величина в скобках называется отступом (margin). Отступ  $i$ -го объекта:

$$M_i = y_i \langle w, x_i \rangle$$

Знак отступа говорит о корректности ответа классификатора, а его абсолютная величина характеризует степень уверенности классификатора в своём ответе.

## 0.15 Что такое точность и полнота? Почему нужно учитывать их вместе?

|                          |                               | Predicted condition  |   |
|--------------------------|-------------------------------|--|---|
|                          |                               | Predicted condition positive (PP)                              | Predicted condition negative (PN)                         |
| Total population = P + N |                               |  |   |
| Actual condition         | Actual condition positive (P) | True positive (TP), hit  | False negative (FN), Type II error, miss, underestimation |
|                          | Actual condition negative (N) | False positive (FP), Type I error, false alarm, overestimation | True negative (TN), correct rejection                     |

Точность (Precision) — доля правильных предсказаний положительного класса

$$\text{Precision} = \frac{\text{True positive}}{\text{True positive} + \text{False positive}}$$

Полнота (Recall) — доля найденных положительных классов

$$\text{Recall} = \frac{\text{True positive}}{\text{True positive} + \text{False negative}}$$

Эти два параметра тесно связаны между собой. Precision показывает насколько можно доверять классификатору в случае срабатывания, а Recall показывает на какой доле истинных объектов классификатор сработал.

Примеры:

- ▷ Первый пример — использование в задаче кредитного scoringa. Пусть в задаче кредитного scoringa ставится условие, что неудачных кредитов должно быть не больше 5%. В таком случае задача является задачей максимизации полноты при условии  $\text{Precision}(a, X) \geq 0.95$ .

- ▷ Второй пример — использование в медицинской диагностике. Необходимо построить модель, которая определяет, есть или нет определенное заболевание у пациента. При этом требуется, чтобы были выявлены как минимум 80% пациентов, которые действительно имеют данное заболевание. Тогда ставят задачу максимизации точности при условии  $\text{recall}(a, X) \geq 0.8$ .

## 0.16 В задаче бинарной классификации доля одного класса составляют 95% выборки. Какие метрики разумно использовать для оценки работы модели? Почему?

Осознаем, что ROC-AUC не зависит от сбалансированности классов, так как строится в осах  $FPR = \frac{FP}{FP+TN}$  и  $TPR = \frac{TP}{TP+FN}$ , то есть доли ответов нормируются на размеры классов, а значит процентное соотношение не задушит нас, тогда как задушит остальные метрики.

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

**ROC** — Receiver Operating Characteristic

$$\text{TPR} = \frac{\text{True positive}}{\text{True positive} + \text{False negative}} = \text{Recall}$$

$$\text{FPR} = \frac{\text{False positive}}{\text{False positive} + \text{True negative}}$$

Отсортируем объекты по вероятности предсказания позитивного/негативного класса, положим их на одну ось. Будем идти скользящей границей (threshold) от минимальной вероятности к максимальной. Считаем 4 характеристики из нашей любимой таблички: FP, FN, TP, TN

TPR и FPR - это наши оси

Полученный график характеристик в данных осях называется ROC-кривой. Посмотрим на ее свойства:

- ▷ Разумный классификатор всегда выше диагонали (диагональ это рандомный)
- ▷ Если кривая строго ниже диагонали меняем знак предсказания
- ▷ Если одна кривая строго выше другой, то соответствующий классификатор лучше

**ROC - AUC** = ROC Area Under Curve - площадь под ROC-кривой: чем лучше классификатор, тем больше (обратное неверно)

## 0.18 Запишите функционал логистической регрессии. Как он связан с методом максимума правдоподобия?

Попробуем предсказать вероятность объекта быть 1-м классом:

$$p_+ = P(y = 1|x) \in [0, 1]$$

Но вероятность живет на отрезке, а нам нужно  $\mathbb{R}$ , тогда нам нужна функция, которая будет имитировать вероятность и иметь значения на  $\mathbb{R}$

$$p_+ = \frac{1}{1 + \exp(-x^T \omega)} = \sigma(x^T \omega)$$

Функционал:  $L_{\text{Logistic}} = \log(1 + \exp(-M_i))$

$$\sigma_\omega(x) = \sigma(x^T \omega)$$

$$\log L(\omega|X, Y) = \log P(X, Y|\omega) = \log \prod_{i=1}^n P(x_i, y_i|\omega)$$

сигмоида симметрична относительно  $(0, 0.5)$

Тогда можем записать следующее:

- ▷ if  $y_i == 1 : P(x_i, 1|\omega) = \sigma_\omega(x_i) = \sigma(M_i)$
- ▷ if  $y_i == -1 : P(x_i, -1|\omega) = 1 - \sigma_\omega(x_i) = \sigma_\omega(-x_i) = \sigma(M_i)$

Тогда получим искомую связь:

$$\log L(\omega|X, Y) = \sum_{i=1}^n \log \sigma(M_i) = - \sum_{i=1}^n \log(1 + \exp(-M_i)) \rightarrow \max$$

## 0.19 Идея метода опорных векторов (в случае разделимой выборки).

Пусть выборка линейно разделима, то есть существует некоторая гиперплоскость, разделяющая классы  $-1$  и  $+1$ . Тогда в качестве алгоритма классификации можно использовать линейный пороговый классификатор:

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

Но для двух линейно разделимых классов возможно много вариантов построения гиперплоскости. Метод опорных векторов выбирает ту гиперплоскость, которая максимизирует отступ между классами:

$$M_i = y_i \langle w, x_i \rangle \quad (\text{считаем что } w_0 \text{ "вшито" в произведение})$$

Чем меньше значение отступа  $M_i$  тем ближе объект к границе классов

Метод опорных векторов (SVM) — один из наиболее популярных методов обучения, который применяется для решения задач классификации и регрессии. Основная идея метода заключается в построении гиперплоскости, разделяющей объекты выборки оптимальным способом. Алгоритм работает в предположении, что чем больше расстояние (зазор) между разделяющей гиперплоскостью и объектами разделяемых классов, тем меньше будет средняя ошибка классификатора.

## 0.20 Опишите жадный алгоритм обучения решающего дерева.

В ходе построения дерева мы разбиваем вашу выборку на две части. Для этого представим, что у нас есть  $L$  и  $R$  — левое и правое поддерево объектов, тогда вспомним, что разбиение в узле строится по критерию вида  $I[x_i^j < b]$ ,  $b$  — порог,  $x_i^j$  —  $j$ -й признак у  $i$ -го объекта. Осталось понять, как выбрать  $j$  и  $b$ .

Для этого будем перебирать все возможные признаки  $j$ , а для каждого признака перебирать все его возможные значения  $x^j$  в выборке, тогда мы можем построить разбиение для каждого из вариантов, осталось выбрать оптимальный. Для этого используют функционал следующего вида:

$$f(X) = \frac{|L|}{|X|} H(L) + \frac{|R|}{|X|} H(R)$$

Осталось определить  $H$ . Как правило, используют одну из следующих двух функций:

- ▷ Критерий Джини. Имеет формулу  $H(X) = \sum_{i=1}^K p_i \cdot (1 - p_i) = 1 - \sum_{i=1}^K p_i^2$ , где  $p_i$  — выборочные вероятности в  $X$  каждого из  $K$  классов.
- ▷ Энтропийный критерий. Имеет формулу  $H(X) = \sum_{i=1}^K p_i \cdot \log p_i$ , где  $p_i$  — выборочные вероятности в  $X$  каждого из  $K$  классов.

Тогда находя минимум  $f(X)$ , выбираем оптимальные  $b$  и  $j$ , разбиваем на два поддерева, рекурсивно повторить.

## 0.21 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.

## 0.22 Если в лист дерева попали объекты разных классов, то какие предсказания нужно выдавать в этом листе? Почему?

Если нет задачи получить вектор вероятностей, то просто наиболее часто встречающийся в листе класс.

Если надо еще получить вектор вероятностей для каждого класса, то вернем вектор из частот встречаемости в листе. Подробнее — надо расписать лагранжиан.

## 0.23 Какое предсказание нужно выдавать в листе дерева в задаче регрессии если мы минимизируем MSE? а в случае MAE?

Дерево в каждом листе предсказывает константу, поэтому будем предсказывать константу оптимальную с точки зрения:

▷ MSE: минимизируем MSE:

$$H(R) = \min_c \frac{1}{|R|} \sum_{(x_i, y_i) \in R} (y_i - c)^2$$

$$c^* = \frac{1}{|R|} \sum_{y_i \in R} y_i \text{ тобишь среднее}$$

▷ MAE: сумма модулей, поэтому предсказываем медиану

## 0.24 Что такое bagging?

Bagging = Bootstrap aggregating — параллельное обучение элементарных классификаторов на бутстрепных (сгенерированных на основе начального датасета путем равномерного распределения) выборках, выбор итогового результата на основе ответов каждого из них (например, абсолютным большинством).

Сгенерируем бутстрепом  $N$  выборок, на каждой обучим модель. Ответом для объекта  $x$  будет являться усредненный ответ каждой модели:  $\frac{1}{N} \sum_{i=1}^N ax_i(x)$

## 0.25 Что такое случайный лес? Чем он отличается от бэггинга над решающими деревьями?

Случайный лес = Random Forest = Bagging + RSM, где RSM = Random Subspace Method — метод, где для каждого сплита выбирается оптимальный признак для каждого дерева для каждого разбиения внутри него лишь из случайного подмножества признаков (эти подмножества генерируем независимо битовой маской).

То есть в результате деревья в наборе становятся непохожими друг на друга из-за случайности в сплитах. Далее, применяем bagging к набору обученных деревьев.

Таким образом

1. Делаем  $M$  бутстрэпированных датасетов.

2. На каждой из них обучаем по одному дереву, но на этапе построения **каждого** разбиения в дереве независимо выбираем подмножество признаков, по которому будет искааться оптимальное разбиение (это нормально, если в итоге лучший признак не попадет туда).

## 0.26 Как в градиентном бустинге обучаются базовые алгоритмы?

Disclaimer: Базовые алгоритмы учатся сообща, помогая друг другу, то есть последовательно, уменьшая при этом ошибку, сделанную предыдущими.

Как? Градиентным спуском в пространстве моделей. Мы двигаемся от модели к модели по антиградиенту функции ошибки. Для этого, очевидно, нам понадобится дифференцируемость этой функции.

Подробнее, пусть имеется модель  $\hat{f}_T(X) = \sum_{i=1}^T \rho_i f_i(X)$ , тогда мы можем для данной модели посчитать градиент лосса на объектах выборки. Давайте  $(T+1)$ -й алгоритм обучать не под выборку, а как раз под эти градиенты (в некотором смысле шаг градиентного спуска, но спускаемся путем обучения модели под градиент). Тогда найдем оптимальную модель  $f_{T+1}(X)$  и оптимальное  $\rho_{T+1}$ . Отсюда получаем, что наш новый алгоритм имеет вид:

$$\hat{f}_{T+1}(X) = \sum_{i=1}^{T+1} \rho_i f_i(X)$$

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

В нейронной сети Loss считается в самом конце, а обучать мы хотим все слои, значит нам надо научиться прокидывать градиент сквозь все слои вплоть до самого первого. Мы можем это делать, так как знаем, что все слои дифференцируемы, тогда

$$\frac{\partial L}{\partial \theta_1} = \frac{\partial L}{\partial f_2(\theta_2)} \cdot \frac{\partial f_2(\theta_2)}{\partial f_1(\theta_1)} \cdot \frac{\partial f_1(\theta_1)}{\partial \theta_1} = \dots = \frac{\partial L}{\partial f_N(\theta_N)} \prod_{i=1}^N \frac{\partial f_i(\theta_i)}{\partial f_{i-1}(\theta_{i-1})} \cdot \frac{\partial f_1(\theta_1)}{\partial \theta_1}$$

В данной формуле подразумевается, что у нас  $N$  слоев, каждый параметризован параметрами  $\theta_i$ , например, для линейной регрессии это вектор весов.

Производная вектора  $y \in \mathbb{R}^n$  по вектору  $x \in \mathbb{R}^m$  — это матрица

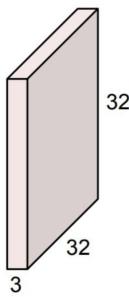
$$\frac{dy}{dx} = \left( \frac{dy_i}{dx_j} \right) \Big|_{i,j=1}^{m,n}$$

## 0.28 Опишите принцип работы свёрточного слоя (CNN).

Пусть имеется какая-то большая матрица данных  $x$  (тензор  $W \times H \times C$ ), наложим на каждый участок этой матрицы фильтр  $w$  (тензор  $F \times S \times C$ ) — окошко, значения которого умножаются на соответствующие значения участка наших данных. От куска матрицы мы получаем всего одно значение после фильтрации:  $w^T x + b$ , в итоге сдвига окошка — новую (свёрнутую) матрицу данных (на самом деле тензор  $(W - F + 1) \times (H - S + 1) \times 1$ ), которая отражает меру «похожести» данных на фильтр. Внимание: число каналов в фильтре и данных совпадает. Так, мы сгенерировали 1 признаковое представление. Но мы хотим знать больше: поэтому заведем несколько независимых фильтров, работающих по отдельности. В итоге получим новый тензор данных из нескольких слоёв-результатов фильтрации, у которых число измерений совпадает с исходным

На вход приходит трехмерный тензор, характеризующий изображение:

32x32x3 image



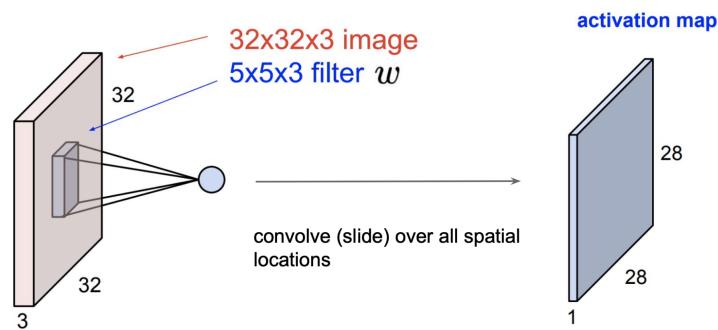
К этому изображению применяется операция свертки, для этого используется определенный фильтр – ядро свертки. Требование к нему такое чтобы глубина совпадала с глубиной входного тензора. Пространственные размеры меньше (у исходного тензора был размер 32x32, а у фильтра 5x5).

5x5x3 filter



**Convolve** the filter with the image  
i.e. “slide over the image spatially,  
computing dot products”

Прикладываем этот фильтр ко всем квадратикам 5x5 исходной картинки. Получим:



## 0.29 В чем недостатки полносвязных нейронных сетей какая мотивация к использованию свёрточных?

1. У полносвязных нейронных сетей очень много обучаемых параметров, тогда как у сверточного слоя из меньше. Пример, у нас есть слой вида  $32 \times 32 \times 3$ , хотим получить тензор  $32 \times 32 \times 10$ . Для сверточного слоя это будет порядка  $10 \cdot (5 \cdot 5 \cdot 3 + 1) = 760$  параметров, а для полносвязного  $> 3^5 \cdot 10^5$ .
2. Полносвязный слой искренне верит в независимость пикселей, тогда как это неправда, нам важно их взаимное расположение.

## 0.30 Опишите принцип работы базового рекуррентного слоя (RNN).

У нас есть контекст  $c$  – вектор фиксированного размера. Далее нам приходит новое слово, кодируем его как-то, то есть получаем вектор  $x$ . Конкатенируем  $c$  с  $x$ , но размер такого вектора может банально не влезть в размер контекста, а значит используем линейное преобразование  $W[x + c]^T + b$ , которое переводит из  $\mathbb{R}^{|c|+|x|}$  в  $\mathbb{R}^{|c|}$ . Далее ставим функцию активации, которая ограничена, например, сигмоида. Она позволяет понять, что запомнить (главное), а что забыть (не главное).

### 0.31 Что такое dropout?

На каждом шаге обучения будем выключать случайные нейроны (домножением выхода на 0 с некоторой вероятностью), чтобы была меньшая зависимость от выхода какого-то конкретного нейрона из прошлого слоя. На тесте включаем все нейроны. Тогда получается что одновременно работает несколько меньших сетей.

Важно понимать, что нейроны не выбрасываются из сети. Их только отключают на этом шаге. Отключение это домножение результата на 1, то есть на них не происходит обучения, так как до них долетает нулевой градиент.

Вспомним про нормализацию.

Так как при обучении на каждом шаге обучалась только половина сети, то полученная оценка описывает менее сильные закономерности, но зато она более устойчивая.

### 0.32 Как dropout и batch normalization меняют свое поведение при эксплуатации модели (в режиме inference)?

Batch Normalization: В режиме train надо обязательно нормировать данные от слоя к слою, потому что каждый раз слой должен ждать на вход одно и то же распределение, иначе ему станет плохо, поэтому есть движущаяся медиана и движущееся среднее, которые обновляются от слоя к слою. В режиме test он нормализует по значениям, которые нашел в train.

Про Dropout: Аналогично нужно “починить” dropout, потому что когда мы после train’а включим режим inference у нас резко включатся совершенно все нейроны, которые изменят распределение. Поэтому надо нормировать выходы, умножая на вероятность dropout’а, чтобы не изменилась дисперсия.

### 0.33 Запишите постановку задачи в методе главных компонент.

Имеется матрица  $X$  размера  $m \times k$  ранга  $N$ . Хотим найти такую матрицу  $\hat{X}$  того же размера, но ранга  $K < N$ , которая минимизирует норму Фробениуса между ними (сумма квадратов элементов).

Таким образом,  $\hat{X} = \arg \min_{Y \in \text{Mat}_{m \times k}(\mathbb{R}): rk Y = K < N} \|X - Y\|_F^2$

# 1 Machine Learning problem statement. Regression, Classification, examples.

В общем случае, чтобы решать задачу, нам надо её поставить. И первое чему должен научиться человек, который занимается машинным обучением, статистикой или управлением командой - это ставить задачу. Давайте ставить задачу машинного обучения корректно.

Что такое supervised?

Давайте предположим, что таргета у нас нет, а есть только данные, которые описаны какими-то признаками.

Можем ли мы сделать что-то с этими данными? Мы можем взять наши данные и даже не имея никакого таргета попытаться из них вытащить какое-то знание: построить какие-то картинки и зависимости, попроверять на них гипотезы, попытаться поделить их на группы. То есть мы можем извлечь знание из данных даже не имея целевой величины, которую нам надо предсказывать. Например: найти внутреннюю структуру в данных. Есть отличие между supervised и unsupervised режимах (обучение с учителем и обучение без учителя) - наличие или отсутствие соответственно таргетов.

Далее мы будем говорить об обучении с учителем. У нас есть некоторая выборка, и она состоит из объектов и их признаков, и каких-то целевых значений, которые им соответствуют.

Training set  $\mathcal{L} = \{x_i, y_i\}_{i=1}^n$  ( $x$  – признак,  $y$  – таргет), где

- ▷  $(x \in \mathbb{R}^p, y \in \mathbb{R})$  для регрессии
- ▷  $x_i \in \mathbb{R}^p, y_i \in \{0, 1\}$  для бинарной классификации
- ▷  $x_i \in \mathbb{R}^p, y_i \in \{c_1, \dots, c_n\}$  для многоклассовой классификации

Если мы говорим про регрессию, то почему, если мы можем предсказать 10 чисел, то у нас таргет может лежать именно в одномерном множестве действительных чисел? Что такое число - это точка на числовой прямой. Что такое точка в 10-мерном пространстве - это радиус-вектор, описывающийся 10 числами. Так что да, мы можем предсказывать хоть 10 чисел одновременно и тогда предсказываемый таргет будет из  $\mathbb{R}^{10}$ . Например: цвет - чтобы его точно предсказать, надо будет предсказать сразу три канала R, G, B, а не один, так как если по отдельности предсказывать каждый канал, то у вас нужный цвет не получится ну никак.

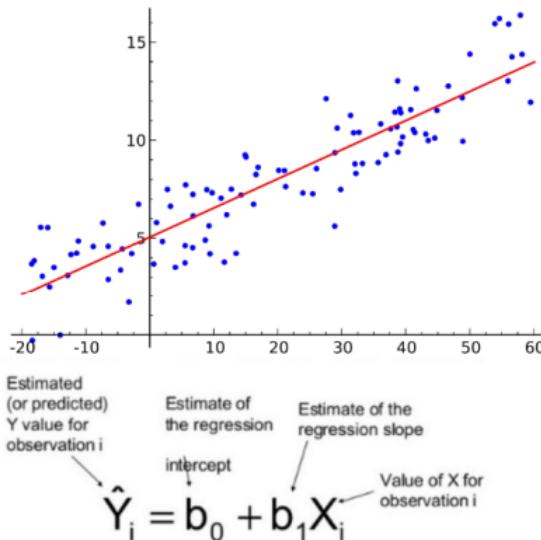
Далее для постановки задачи машинного обучения должна быть модель  $f(x)$ , которая предсказывает для каждого объекта какую-то величину. И конечно же, нужна функция потерь  $Q(x, y, f)$ , которую мы используем чтобы оценить, какая модель лучше работает на наших данных. **Пока вы не знаете как мерить качество в вашей задаче, вы её не поставили и вы не знаете как её решать.** Очень часто люди в стартапах и индустрии говорят нам нужна в первую очередь выборка, так вот пока вы не знаете как будет у вас выглядеть функция потерь и какая будет гипотеза, вам не надо собирать выборку. Потому что в таком случае вы не понимаете какую задачу вы решаете, и пока вы задачу доформулируете, ваши ожидания к данным могут измениться.

Где мы будем минимизировать функцию потерь?

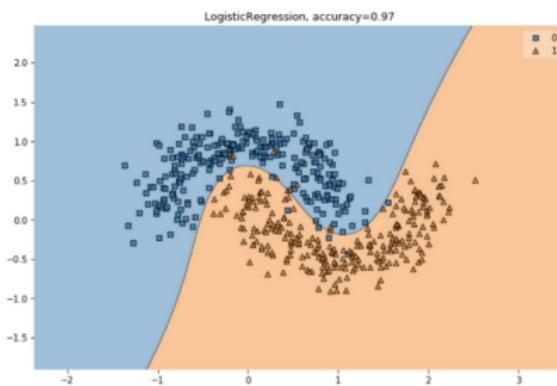
Выборку мы уже собрали и ничего сделать с ней не можем,  $x, y$  - зафиксированы. Модель  $f$  скорее всего зависит от каких то параметров, поэтому минимизировать мы будем по параметрам модели, или хотя бы найти оптимальную функцию  $f$ , если их какое-то семейство.

**Примеры задач регрессии и классификации по имеющимся точкам в пространстве признаков:**

Проводим по точкам прямую, наиболее хорошо фитирующую точки



Проводим гиперплоскость, наиболее хорошо разделяющую все объекты на 2 класса



## 2 How to measure quality in classification: accuracy, balanced accuracy, precision, recall, f1-score, ROC-AUC, multiclass extensions

Метрики качества в задачах классификации

### 1. Accuracy

Accuracy - доля правильных ответов при классификации.

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

Accuracy бесполезна в задачах с неравными классами

**Пример** Допустим, мы хотим оценить работу спам-фильтра почты. У нас есть 100 не-спам писем, 90 из которых наш классификатор определил верно (True Negative = 90, False Positive = 10), и 10 спам-писем, 5 из которых классификатор также определил верно (True Positive = 5, False Negative = 5). Тогда:  $\text{accuracy} = \frac{5+90}{5+90+10+5} = 86.4$ . Однако если мы просто будем предсказывать все письма как не-спам, то получим более высокую accuracy:  $\frac{0+100}{0+100+0+10} = 90.9$

### 2. Balanced accuracy

Также есть ещё balanced accuracy. Когда мы знаем, что нам важны все классы, но мы знаем, что у одного класса маленький размер. В таком случае мы можем сбалансировать и посчитать accuracy

для каждого класса, и уже полученные accuracy (их будет столько сколько классов) усреднить между собой. Но в реальной жизни такая метрика оказывается довольно бесполезной, так как в такой агрегации нужно смотреть ещё на дисперсию, потому что повышая среднее, можно повысить еще и дисперсию (то есть у кого-то густо, у кого-то пусто). Проблему дисбаланса классов такая метрика решает не очень хорошо.

$$\text{balanced accuracy} = \frac{1}{\# \text{classes}} \sum_{\# \text{classes}} \text{accuracy}_i$$

#### 4. 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

|                  |                          | Predicted condition  |   |
|------------------|--------------------------|--|---|
|                  |                          | Predicted condition positive (PP)  | Predicted condition negative (PN)   |
| Actual condition | Total population = P + N | Actual condition positive (P)  | Actual condition negative (N)   |
|                  |                          | <b>True positive (TP), hit</b><br><b>False negative (FN), Type II error, miss, underestimation</b> | <b>False positive (FP), Type I error, false alarm, overestimation</b><br><b>True negative (TN), correct rejection</b> |

$$\text{precision} = \frac{TP}{TP + FP}$$

#### 5. Recall

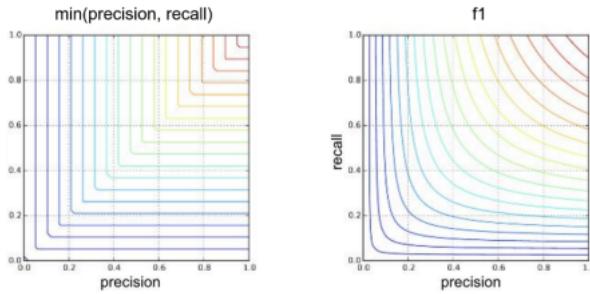
Recall - полнота. (количество сбитых самолётов / общее количество самолётов). Количество правильных положительных ответов / каким должно было быть количество положительных ответов.

$$\text{recall} = \frac{TP}{TP + FN}$$

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

#### 6. F-score

У нас есть две принципиально разных метрики - precision и recall, мы хотим минимизировать и ошибку 1 рода, и ошибку 2 рода. Поэтому нам придется рассматривать то одну, то другую, как-то по очереди минимизировать, это неудобно рассматривать две метрики одновременно. Соединим в себе две метрики, и будем брать минимум из них. Чуть-чуть подняв recall, мы чуть-чуть поднимаем precision. Такая метрика не дифференцируема



поэтому нам нужна более гладкая функция. Так появилась F1-score - гармоническое среднее между precision и recall. Линии уровня имеют разный цвет, цветом отмечены различные значения f1-score. В правом верхнем углу - идеальная ситуация.

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

Как сделать так, чтобы важность precision и recall можно было пофиксить? Например, вы решаете задачу машинного обучения в медицине, и вы хотите выделить людей, которые имеют страшную болезнь. Вам выгоднее с точки зрения человеческих жизней, пресказать лучше должно человеку, что он болен, чем пропустить человека и сказать больному, что он здоров. Таким образом recall для нас становится важнее в этом примере. Чтобы использовать F-score надо сдвинуть баланс, чтобы этот баланс соблюсти, у нас появляется параметр  $\beta$ .

$$F_\beta = (1 + \beta^2) \frac{precision \cdot recall}{\beta^2 precision + recall}$$

Член  $1 + \beta^2$  нам нужен для нормировки, чтобы метрика оставалась между 0 и 1. Чтобы у нас был важнее recall мы делаем  $\beta < 1$  в знаменателе. В числителе обе метрики равнозначны.

## 7. 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).

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

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

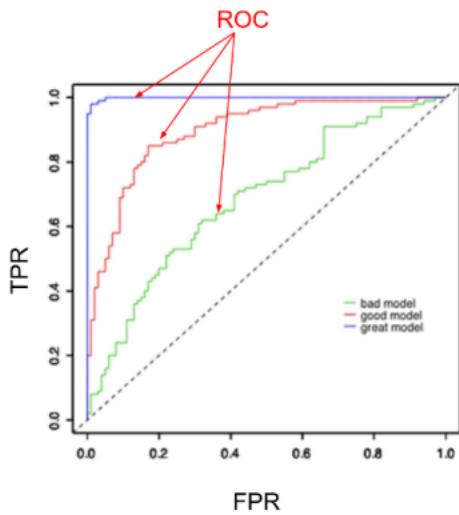


Рис. 20: ROC-кривые для плохого, хорошошего и замечательного классификаторов.

Главные свойства ROC кривой:

1. Baseline является случайным предсказанием, и лежит на линии  $y = x$  (диагонали на графике).
2. Нормальный классификатор, который предсказывает что-то разумное, будет выше диагонали. Понятно, что любой классификатор хороший, лучше чем случайное угадывание.
3. Если предсказание лежит ниже baseline, то надо просто поменять знак, и тогда получится хороший классификатор, кривая отразится относительно  $y = x$ .
4. Если график ROC одного классификатора строго выше (во всех точках) графика другого классификатора, то тот классификатор, который выше - лучше.
5. Количество отсечек (ступенек) на кривой равно количеству объектов (не больше его). Если сетка установлена чаще, чем количество объектов, вы не получите ничего интересного

### Площадь под кривой ROC.

Всё согласуется с тем, что мы постулировали, что чем выше кривая ROC, тем лучше классификатор - площадь под кривой тоже будет больше. Тогда сложную кривую мы сведем к 1 числу. По этому числу уже можно ранжировать модели между собой.

1. Лежит в промежутке  $[0,1]$ , но эффективно в  $[0.5,1]$ , т.к. baseline имеет ROC-AUC = 0.5.
2. Из того, что кривая лежит в каждой точке выше другой кривой, следует, что ROC-AUC у неё выше.

Но наоборот не следует! Бывают случаи, когда один классификатор лучше в одной области, а другой в другой области, а ROC-AUC у них равны.

## 8. Мультиклассовые метрики

Все предыдущие метрики были введены для бинарной классификации, чтобы перейти к метрикам мультиклассовой классификации, требуется взять среднее в каком-то смысле от метрик для всех бинарных классификаторов в мультиклассовой задаче. Буква S - сэмпл, то есть вся выборка, а буква L - классы. В таблице показаны варианты мультиклассовых метрик из библиотеки sklearn.

| average    | Precision   | Recall  | $F_\beta$   |
|------------|---|---|---|
| “micro”    | $P(y, \hat{y})$   | $R(y, \hat{y})$   | $F_\beta(y, \hat{y})$   |
| “samples”  | $\frac{1}{ S } \sum_{s \in S} P(y_s, \hat{y}_s)$                                    | $\frac{1}{ S } \sum_{s \in S} R(y_s, \hat{y}_s)$                                    | $\frac{1}{ S } \sum_{s \in S} F_\beta(y_s, \hat{y}_s)$                                    |
| “macro”    | $\frac{1}{ L } \sum_{l \in L} P(y_l, \hat{y}_l)$                                    | $\frac{1}{ L } \sum_{l \in L} R(y_l, \hat{y}_l)$                                    | $\frac{1}{ L } \sum_{l \in L} F_\beta(y_l, \hat{y}_l)$                                    |
| “weighted” | $\frac{1}{\sum_{l \in L}  \hat{y}_l } \sum_{l \in L}  \hat{y}_l  P(y_l, \hat{y}_l)$ | $\frac{1}{\sum_{l \in L}  \hat{y}_l } \sum_{l \in L}  \hat{y}_l  R(y_l, \hat{y}_l)$ | $\frac{1}{\sum_{l \in L}  \hat{y}_l } \sum_{l \in L}  \hat{y}_l  F_\beta(y_l, \hat{y}_l)$ |

### 3 How to measure quality in regression: MSE, MAE, R2.

#### 1. MSE

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

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

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

#### 2. MAE

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

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

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).

#### 3. $R^2$

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

Член вычитаемый из 1 можно интерпретировать как оценённую дисперсию, отнесённую к реальной дисперсии. Ещё можно сказать, что мы модель сравниваем с моделью, которая предсказывает просто константу. Смотрим насколько далеко ушла наша модель от тупого предсказания среднего. Может иметь отрицательные значения, так как числитель может неограниченно расти.

#### MAPE

$$MAPE = \frac{1}{n} \sum_{i=1}^n \left| \frac{Y_i - \hat{Y}_i}{Y_i} \right|$$

#### SMAPE

Когда есть большие выбросы в MAPE, следует использовать отнормированную и на прогнозируемое значение метрику.

$$SMAPE = \frac{1}{n} \sum_{i=1}^n \frac{2|Y_i - \hat{Y}_i|}{|Y_i| + |\hat{Y}_i|}$$

## 4 Maximum likelihood estimation, how is it related to regression and classification

Функция правдоподобия это ни что иное как условная вероятность выборки, при условии что параметрическое семейство, которое описывает эту выборку обладает ровно таким параметром.

$$L(\theta|X, Y) = P(X, Y|\theta)$$

От правдоподобия мы хотим только одну вещь - максимизировать. Хотим наиболее правдоподобные параметры при условии нашей выборки.

$$L(\theta|X, Y) \rightarrow \max_{\theta}$$

Так как матрица признаков имеет независимые объекты i.i.d, мы можем расписать условную вероятность как произведение условной вероятности по каждому объекту.

$$L(\theta|X, Y) = P(X, Y|\theta) = \prod_i P(x_i, y_i|\theta)$$

Argmax функции совпадает с argmax любого монотонного преобразования над функцией. Тут можно использовать логарифм, который из произведения делает сумму, логарифм - монотонное преобразование, поэтому мы можем заменить произведение на сумму логарифмов условной вероятности и его уже максимизировать:

$$\log L(\theta|X, Y) = P(X, Y|\theta) = \sum_i \log P(x_i, y_i|\theta) \rightarrow \max_{\theta}$$

**Замечание 4.1.** ФУНКЦИЯ ПРАВДОПОДОБИЯ - ЭТО НЕ РАСПРЕДЕЛЕНИЕ НАД ПАРАМЕТРАМИ  $\theta$

**Замечание 4.2.**

Большинство моделей работает по принципу максимального правдоподобия, линейная регрессия и классификация работают именно по нему.

В задаче линейной регрессии мыходим минимизировать MSE для получения оптимального вектора весов. Задача минимизации MSE эквивалентна методу максимального правдоподобия при определенных условиях по теореме Гаусса-Маркова.

В задаче классификации мы напрямую используем MLE для выбора параметров

## 5 Naive bayesian classifier, how does it work

Наивное предположение: фичи независимы.

Вспомним теорему Байеса, которая дает нам формулу условной вероятности:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Ставится задача  $K$ -классовой классификации (то есть каждый элемент по какому-то признаку может быть одного из  $k$  возможных классов).

$$x_i \in \mathbb{R}^p, y_i \in \{C_1, \dots, C_k\}$$

Давайте прикинем условную вероятность метки класса при условии того, что объект описывается какими-то фичами.

$$P(y_i = C_k | x_i) = \frac{P(x_i | y_i = C_k) P(y_i = C_k)}{P(x_i)}$$

То есть мы оцениваем, какова вероятность того, что метка класса это  $C_k$  при условии, что мы наблюдаем  $x_i$ .

Уже имея данную формулу мы можем оценивать вероятность лейблов.  $P(y_i = C_k)$  оценим как частоту (встречаемости) каждого класса в нашей выборке (просто руками посчитаем, насколько вероятно попадание в тот или иной класс). Но нам надо оценить  $P(x_i | y_i = C_k)$ . Это сделать непонятно, как, потому что  $x_i$  – это вектор. Здесь и понадобится наше *наивное* предположение – все признаки независимы и тогда хорошо факторизуются (мы считаем, что признаков у нас  $p$ ).

$$P(x_i | y_i = C_k) = \prod_{n=1}^p P(x_i^n | y_i = C_k)$$

В реальной жизни признаки часто зависимы, причем они бывают зависимы довольно сложным образом, так что просто так взять и подчистить данные не удастся. Но есть хорошая новость – признаки бывают частично независимы (например, среди них бывают независимые подмножества), так что какой-то смысл в этом есть. Теперь вектор  $x_i$  представляется как вектор  $p$  независимых случайных величин. Мы хотим найти метку класса, которая наиболее вероятна

$$C^* = \underset{k}{\operatorname{argmax}} P(y_i = C_k | x_i) = \underset{k}{\operatorname{argmax}} \frac{P(x_i | y_i = C_k) P(y_i = C_k)}{P(x_i)}$$

Это и будет нашим предсказанием.

Теперь заметим, что знаменатель не зависит от  $k$ , поэтому нахождение максимума он не влияет. То есть если его убрать, у нас, конечно, выражение перестанет быть вероятностью, но метка класса, при котором достигается этот самый максимум, не изменится. Поэтому на него можно забыть, и даже сэкономить на этом какое-то время (хотя и не очень большое).

Правда, на самом деле, мы в любом случае его и так практически посчитаем по пути, так как  $P(x_i) = \sum_k P(x_i | y_i = C_k)$ .

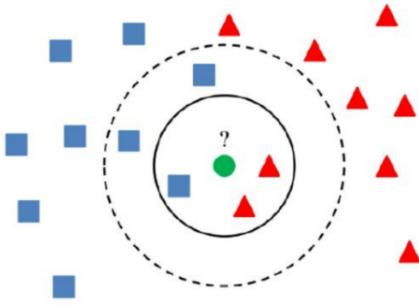
Для того, чтобы посчитать  $P(x_i^l | y_i = C_k)$ , требуется ввести какое-то распределение на наши признаки. Строго понять это вряд ли получится, так что или нам это было известно, или мы должны просто предположить, из какого распределения они пришли. Тут есть 2 пути: 1 - если они тоже все категориальные, можем просто посчитать частоту каждого из них, 2 - если они какие-то числовые, то тут чуть сложнее, нам придется ввести какое-то априорное распределение на  $x$ . Самый простой случай, когда  $x$  пришел из нормального распределения. В этом случае мы считаем дисперсию и среднее по выборке, предполагая что распределение нормальное, и тогда можем оценить вероятность.

Такой классификатор будет хорошо работать на классификации текстов (например, определять, грубый ли комментарий в соц.сети).

## 6 K-nearest neighbours classifier, how does it work

### Интуиция и идея kNN

**Пример 6.1.** Рассмотрим картинку. Пусть у нас все объекты двух типов – синий квадрат или красный треугольник. И вот нам поступил объект, который на картинке отмечен зеленым кругочком. Мы про него хотим понять, кто он на самом деле (к какому из двух классов относится). Это несложно делается с помощью метода ближайших соседей.



Собственно, плюс метода ближайших соседей как раз в том, что он абсолютно интуитивный. Идея такая: «чем ближе я к другим объектам в признаковом пространстве, тем больше я на них похож». Мы можем использовать расстояние между объектами в пространстве признаков как некоторую меру похожести.

Допустим на картинке мы будем считать расстояние по  $L_2$ -норме, то есть использовать Евклидово расстояние, к которому мы привыкли. В этой и других задачах в принципе можно ввести любую другую метрику и по ней посчитать расстояние.

Если у нас просто есть выборка, и она желательно задана в каком-то линейном метрическом пространстве, то, как правило, там мало категориальных фичей, этим можно воспользоваться и применить собственно описанную выше идею. Вот тут появляется выбор гиперпараметра модели: тут это параметр  $k$  – сколько ближайших соседей мы выбираем для того, чтобы решить, к какому классу относится объект. Если мы возьмем одного соседа, он может оказаться выбросом из выборки и дать нам неверный результат. Но если мы возьмем информацию о нескольких соседях, мы будем более уверены в предсказании класса объекта.

**Пример 6.2** (Пример из жизни, поясняющий интуицию:). Так, например, если вокруг вас вечером много физтехов, то вы, скорее всего, студент физтеха и находитесь в общежитии.

## Алгоритм kNN

В итоге получаем такой алгоритм:

1. Берем датасет и запоминаем координаты объектов в пространстве признаков.
2. Берем новое наблюдение и считаем все расстояния до каждого объекта в датасете.
3. Выберем  $k$  объектов с минимальным расстоянием до вашего нового наблюдения.
4. Выбираем класс, который доминирует среди этих  $k$  соседей. Он-то и будет нашим предсказанием для рассматриваемого объекта.

Есть минусы в таком подходе:

1. При разных значениях гиперпараметра числа соседей может получаться различный результат. Можем проверять качество работы алгоритма при фиксированном параметре, но не можем сходу оптимизировать его (оптимизируется только перебором, к сожалению).
2. В настоящее время при больших выборках занимает большое вычислительное время из-за подсчета большого количества расстояний (а в пространствах с большой размерностью расстояния считаются между длинными векторами, что тоже долго, все ведь помним плюсы с длинной арифметикой :)).

## Взвешенный kNN

Теперь подумаем, как еще мы можем предсказывать класс на основе информации о  $k$  ближайших соседях. Сейчас мы выбираем самый часто встречающийся класс, но, может быть, можно делать лучше? Есть такой выход.

Он называется взвешенным  $kNN$ . В этой вариации мы присваиваем вес лейблу каждого соседа в зависимости от расстояния от нашего объекта до соседа. То есть чем дальше сосед, тем меньший вклад в ответ он вносит.

## Как с помощью $kNN$ решить задачу регрессии?

Практически ничего не меняется, только таргетами станет континуальная величина, то есть числа. Что надо сделать с матрицей признаков перед тем как использовать на ней  $kNN$ ?

Ее обязательно надо отнормировать.

**Пример 6.3.** Если в одной матрице есть признак в сантиметрах и признак в метрах, то это странно. Или в бухгалтерии зарплата представлена в матрице в рублях и в копейках как два разных признака, то по-хорошему их надо объединить в один признак или привести в одну шкалу, то есть чтобы были рубли и доли рублей.

Если мы ничего не знаем о природе нашей выборки, то самое лучшее, что мы можем с ней сделать, это отнормировать ее в промежуток  $[0; 1]$ , то есть вычесть минимум и поделить на максимум.

## 7 Linear regression. Problem statement for the MSE loss function case. Analytical solution. Gauss-Markov theorem. Gradient approach in linear regression.

### Постановка задачи

Пусть задан датасет  $\mathcal{L} = \{x_i, y_i\}_{i=1}^N$ , где  $x_i \in \mathbb{R}^n, y_i \in \mathbb{R}$ . Наша модель линейная, то есть предсказание выглядит следующим образом:

$$\hat{y} = w_0 + \sum_{k=1}^p x_k \cdot w_k = x^T w$$

Чтобы отдельно не записывать свободный член, добавим в  $x$  признак, тождественно равный единице. Тогда соответствующий коэффициент в  $w$  будет отвечать за свободный член:

$$(1 \ x_1 \ \dots \ x_p) \begin{pmatrix} w_0 \\ w_1 \\ \vdots \\ w_p \end{pmatrix} = w_0 + \sum_{k=1}^p x_k \cdot w_k$$

Задача оптимизации, которую мы решаем – это минимизация какого-то функционала. Например, среднеквадратичной ошибки (такой подход называется методом наименьших квадратов):

$$\hat{w} = \arg \min_w \|Y - \hat{Y}\|_2^2 = \arg \min_w \|Y - Xw\|_2^2$$

В случае минимизации среднеквадратичной ошибки существует аналитическое решение:

$$\begin{aligned} \|Y - Xw\|_2^2 &= \langle Y - Xw, Y - Xw \rangle = Y^T Y - (Xw)^T Y - Y^T (Xw) + (Xw)^T Xw = \\ &= Y^T Y - w^T X^T Y - Y^T Xw + w^T X^T Xw \end{aligned}$$

$$\nabla_w \|Y - Xw\|_2^2 = 0 - X^T Y - X^T Y + (X^T X + X^T X) w$$

Приравнивая  $\nabla_w \|Y - Xw\|_2^2$  к нулю, получаем:

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

**Замечание 7.1.** Аналитическое решение существует только для 2-нормы. Для остальных норм решение ищется только градиентными методами.

**Теорема 7.2** (Гаусса-Маркова). Пусть целевые значения выражаются в следующем виде:

$Y = Xw + \varepsilon$ , где  $\varepsilon = (\varepsilon_1, \dots, \varepsilon_N)$  – случайный вектор, при этом:

1.  $\mathbb{E}\varepsilon_i = 0 \forall i$
2.  $\mathbb{D}\varepsilon_i = \sigma^2 < \infty \forall i$
3.  $\text{cov}(\varepsilon_i, \varepsilon_j) = 0 \forall i \neq j$

Тогда решение задачи наименьших квадратов  $\hat{w} = (X^T X)^{-1} X^T Y$  является оптимальным среди несмешённых.

Другими словами,  $\hat{w}$  является несмешённой ( $\mathbb{E}\hat{w} = w$ ) и оптимальной (то есть имеет наименьшую дисперсию среди всех несмешённых).

### Основные функции потерь в регрессии

$$\text{MSE}(y, \hat{y}) = \frac{1}{N} \|y - \hat{y}\|_2^2 = \frac{1}{N} \sum_i (y_i - \hat{y}_i)^2$$

- ▷ Применима теорема Гаусса-Маркова
- ▷ Дифференцируема
- ▷ Чувствительна к шуму

$$\text{MAE}(y, \hat{y}) = \frac{1}{N} \|y - \hat{y}\|_1 = \frac{1}{N} \sum_i |y_i - \hat{y}_i|$$

- ▷ Не дифференцируема (можно доопределить производную в нуле нулём)
- ▷ Более устойчива к шуму

## 8 Regularization in linear models: L1 и L2, their properties. Probabilistic interpretation.

Напомним, что аналитическое решение для метода наименьших квадратов выглядит следующим образом:

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

Видно, что нам приходится искать матрицу, обратную к  $X^T X$ . Однако, что делать, если эта матрица вырождена (или близка к вырожденной)?

Сначала подумаем, что означает вырожденность матрицы  $X^T X$ . Это означает, что в этой матрице (а значит, и в матрице  $X$ ) есть линейно зависимые столбцы (признаки). В таком случае

столбец весов не может быть определён однозначно, так как будет фиксирована только сумма весов этих параметров, а сами параметры могут определяться континуальным количеством способов.

Если же матрица  $X^T X$  близка к вырожденной, то решение получается *нестабильным* (то есть при добавлении небольшого шума в данные вектор весов меняется сильно).

Чтобы решить эту проблему, нам хочется ограничить свободу выбора этих коэффициентов, наложить дополнительное условие, чтобы решение стало единственным. Давайте потребуем, чтобы норма вектора весов была наименьшей из возможных. Тогда задача оптимизации запишется следующим образом:

$$\hat{w} = \underset{w}{\operatorname{argmin}} (\|Y - Xw\|_2^2 + \lambda \|w\|_2^2)$$

Коэффициент  $\lambda$  – это гиперпараметр, который отвечает за то, насколько важно для нас, чтобы норма вектора весов была маленькая.

**Важное замечание.** В слагаемом  $\lambda \|w\|_2^2$  мы записываем  $w$  без свободного члена ( $w_0$ ), так как иначе наша модель будет стараться уменьшать в том числе и  $w_0$ , то есть пытаться провести гиперплоскость через ноль. В общем случае это не следует из каких предположений, поэтому накладывать ограничение на  $w_0$  нельзя.

Аналитическое решение такой задачи будет выглядеть следующим образом:

$$\hat{w} = (X^T X + \lambda I)^{-1} X^T Y$$

Видно, что теперь обратная матрица существует всегда (так как к  $X^T X$  мы добавляем единичную матрицу).

Подход, при котором мы накладываем дополнительные ограничения, называется *регуляризацией*. При этом не обязательно накладывается ограничение на норму вектора весов (см. другие примеры в билете 27).

Можно применять не только  $L2$  регуляризацию, но и  $L1$ .

## L2

- ▷ имеет аналитическое решение
- ▷ дифференцируема

## L1

- ▷ не дифференцируема
- ▷ "отбирает" признаки

## 9 Logistic regression. Equivalence of MLE approach and logistic loss minimization.

### Постановка задачи

$X \in \mathbb{R}^{n \times p}$ ,  $Y \in C^n$ , где  $C$  – это множество меток классов (конечное). Если метки не упорядочены, то это называется *задачей классификации*.

Пока что будет заниматься бинарной классификацией и для удобства будем считать, что  $C = \{-1, 1\}$ . Обозначим за  $c(X)$  функцию, которая возвращает  $\hat{Y}$ , то есть предсказание классов нашей модели.

Допустим у нас есть матрица признаков и вектор весов, как из этого всего, что уже было в линейной регрессии, составить линейную классификацию? Надо делить на классы предсказанный ответ, обрубая по какой-то границе, если у нас два класса.

$$c(x) = \begin{cases} 1, & f(x) \geq 0 \\ -1, & f(x) < 0 \end{cases}$$

Пусть вектор  $w$  является нормалью к гиперплоскости в  $p$ -мерном пространстве фичей (гиперплоскость задаётся уравнением  $x^T w = 0$ ), с одной стороны гиперплоскости будут одного класса объекты, с другой — другого класса объекта. Тогда  $c(x)$  можно переписать в следующем виде:

$$c(x) = \text{sgn}(f(x)) = \text{sgn}(x^T w)$$

Введём понятие margin (отступ):

$$M_i = y_i f(x_i) = y_i x_i^T w$$

Это некоторый аналог ориентированного расстояния, причём если  $M_i \leq 0$ , значит объект классифицирован неправильно. Тогда логично предложить следующую функцию потерь:

$$\text{Loss} = \sum_{\text{by objects}} [M_i \leq 0]$$

Но у такой loss function будут проблемы, так как это не гладкая функция. Это главная причина почему мы не будем использовать такой loss, по крайней мере в наших текущих задачах. Есть ещё один менее очевидный минус: такая метрика ничего нам не говорит об уверенности классификатора. Представьте объекты как точки в пространстве признаков, есть гиперплоскость классификатора, и было бы разумно полагать, что объекты на границе, где перемешиваются два класса, будут с меньшей уверенностью разделены классификатором, чем лежащие далеко "в толще" класса, которые с большей точностью принадлежат своему классу.

Что можно сделать с имеющейся функцией потерь — приблизить её к гладкой функции. Предлагается использовать логистическую регрессию.

$$p_+ = P(y = 1|x) \in [0, 1]$$

Регрессионная модель живет в пространстве  $\mathbb{R}$ , а вероятность живёт исключительно в промежутке  $[0, 1]$ . Как с этим быть? Применить функцию, которая переводит  $\mathbb{R}$  в промежуток  $[0, 1]$  и таким образом моделлирует вероятность. Используем некий трюк и составим величину, которая уже будет куда ближе к  $\mathbb{R}$ :  $\frac{p_+}{1-p_+} \in [0, +\infty)$

Осталось отразить промежуток  $[0, +\infty]$  в  $\mathbb{R}$ , с чем хорошо справляется функция логарифма:  $\log \frac{p_+}{1-p_+} \in \mathbb{R}$ . Таким образом мы объединили мир классификации и мир регрессии. Теперь запишем предсказание через вектор признаков и вектор весов, а также выразим вероятность объекта иметь класс 1:

$$\frac{p_+}{1-p_+} = \exp(x^T w)$$

$$p_+ = \frac{1}{1 + \exp(-x^T w)} = \sigma(x^T w)$$

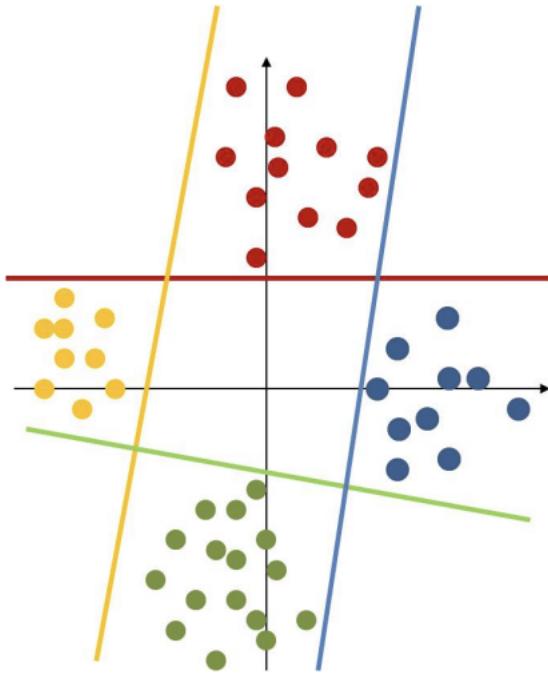
Получилось, что вероятность объекта иметь 1 класс равна сигмоиде от привычного выражения для линейной регрессии.

## 10 Multiclass classification. One-vs-one, one-vs-all, their properties.

Что делать, если мы хотим классифицировать объекты не на два класса, а на большее количество? Есть два разных подхода

**One vs Rest**

Исходя из названия понятен смысл: мы берём один класс, все остальные объекты помечаем другим классом. Далее на этом тренируем классификатор, потом берём следующий класс, кроме этого второго класса, помечаем остальные объекты противоположным классом. И так тренируем столько классификаторов, сколько у нас есть классов.

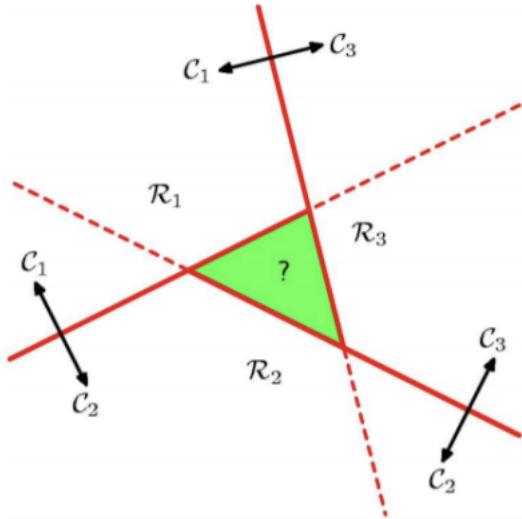


Довольно очевидно, что тот, у кого наибольшая вероятность в таком процессе, предсказывается следующим образом: берётся точка в пространстве, мерим вероятности относительно каждого из построенных классификаторов и смотрим у кого больше вероятность, к тому классу мы и относим. Как должны быть расположены классы в пространстве признаков, чтобы поломать такой классификатор? Центры каждого класса должны быть расположены на одной прямой, такая ситуация поломает подобную классификацию. Понятно, что крайние два класса будут хорошо разделяться, а средний не будет отделяться вообще никак, потому что нельзя линейным классификатором (одной гиперплоскостью) отделиться от двух других классов справа и слева.

Есть ещё одна проблема, которая менее заметна на первый взгляд: серые зоны. На рисунке серую зону можно увидеть в центре. Если в ту область попадают объекты из обучающей выборки, непонятно к какому классу их отнести, у нас нет таких данных. Равно как и в остальных серых зонах по бокам, тоже непонятно что будет происходить. В достаточно простом случае мы можем это предсказать - финальное решение поделит серые зоны между каждыми двумя классификаторами пополам, а в центре на столько частей сколько классов всего. Несовпадение границ проведенных и границ финальных связано с тем, что мы принимаем решение коллективно.

### One vs One

В такой стратегии мы будем сравнивать один класс с каким-нибудь другим классом из имеющихся, применяя опять линейный классификатор.



Буквами  $R_i$  на картинке обозначены кластера тех объектов, которые мы будем классифицировать. А сплошные линии, переходящие в пунктирные это классификаторы. Класс  $c_2$  отделяется классификатором от класса  $c_1$ , несмотря совершенно на класс 3, и тд.

### Сравнение

В одном случае мы тренируем только  $k$  классификаторов, но тренируем их на полном датасете каждый — это отражается на времени обучения всего классификатора. Во втором случае мы тренируем квадратичное число классификаторов, но с другой стороны у нас есть плюс в том, что если классы равномерно распределены, то каждый раз мы будем только два кластера из всех рассчитывать (то есть меньше чем весь датасет). Но в стратегии one vs one есть проблема в том, что слишком мало сэмплов. То есть существуют классы, которых меньше в выборке, и с ними может возникнуть проблема. Надо заметить, что в таком процессе one vs one мы убираем строки из матрицы признаков, а не столбцы, то есть количество признаков сохраняется, а именно количество объектов при обучении уменьшается.

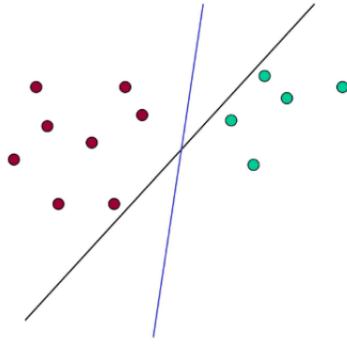
## 11 Support vector machine. Optimization problem for SVM. Kernel trick. Kernel properties.

### Мотивация:

Ранее мы рассматривали логистическую регрессию и log loss в виде логарифма от сигмоиды. Но данную задачу приближения loss function гладкой функцией можно решать и другими способами, один из них - SVM.

Давайте начнём с простого случая, когда у нас есть линейно разделимая выборка (хотя на практике такое вряд ли произойдет). Подумаем, как выбрать оптимальную гиперплоскость, которая разделяет два класса. Вопрос в том, что таких гиперплоскостей может быть очень много (например, какую-нибудь подходящую гиперплоскость можно чуть-чуть или даже не чуть-чуть пошевелить).

Получается, что много моделей представляют решение одной и той же задачи классификатора, и одно и то же значение эмпирического риска. В итоге у нас решение не единственno, а это плохо, потому что непонятно тогда, какое выбирать.



## Идея:

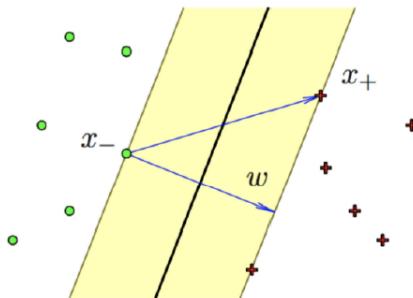
Поэтому давайте поймем, какая модель (гиперплоскость) линейной классификации лучше? Полезно не только строить гиперплоскости, но и максимизировать ширину зазора между классами, которую эта гиперплоскость определяет (собственно, интуитивно кажется, что так разделение будет качественнее всего).

$$\exists w, w_0 : M_i(w, w_0) = y_i(\langle w, x_i \rangle - w_0) > 0 \forall i \in \{1, \dots, l\}$$

Формула выше представлена для ситуации, когда *bias* не включен в вектор весов, и в вектор признаков не включена в начале 1. Поэтому мы вычитаем ещё смещение  $w_0$ .

Если умножить гиперплоскость на константу, наклон не поменяется, поэтому мы можем произвести нормировку (никогда не было, и вот опять) *margin* в 1 с помощью изменения параметров модели  $w$ .

Рассмотрим следующую картинку:



Отступ на ней это значение вектора  $w_0$ . Будем считать, что для самого левого красного плюсика (если смотреть на расстояние до гиперплоскости) у нас верно  $\langle w, x_+ \rangle - w_0 = 1$ . Тогда все точки правее будут иметь  $\langle w, x \rangle - w_0 \geq 1$ , а все точки левее самого правого зеленого кружочка будут иметь  $\langle w, x \rangle - w_0 \leq -1$ .

Мы хотим максимизировать ширину полосы между классами. Тогда давайте максимизировать, вспоминая линейную алгебру и то, как оценить расстояние от точки до прямой. Будем максимизировать расстояние между крайней левой точкой «положительного» класса, и крайней правой точкой «отрицательного» класса.

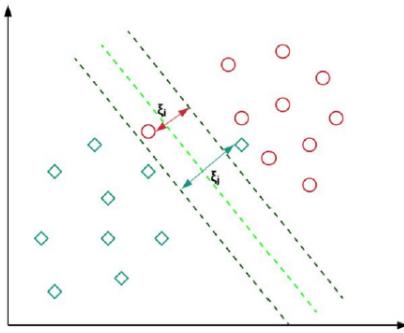
$$\frac{\langle x_+ - x_-, w \rangle}{\|w\|} \geq \frac{2}{\|w\|} \rightarrow \max$$

Широкая полоса между классами хороша тем, что даже имея девиации, классификатор всё ещё будет хорошо предсказывать разделение по классам. То есть если у нас появится в выборке несколько новых объектов, наш классификатор не будет резко менять свое решение, как могло произойти ранее. Потому что теперь мы не просто ищем гиперплоскость, но и максимизируем ширину между классами объектов. Как мы уже сказали ранее, модуль отступа должен быть положителен для каждого класса объектов с учетом нашей нормировки, так как выборка линейно разделима по нашему предположению. Максимизируем модуль расстояния между классами, значит минимизируем квадратичный функционал нормы вектора весов:

$$\begin{cases} \frac{1}{2}\|w\|^2 \rightarrow \min_{w, w_0} \\ M_i(w, w_0) \geq 1 \quad i = 1, \dots, l \end{cases}$$

Всё было бы здорово и задача оптимизации была бы поставлена, но мы сделали одно очень далекое от правды предположение, что наша выборка линейно разделима. На практике есть шум, сложные зависимости, которые не обязательно линейно разделяются. Или объекты одного класса могут попасть в глубину объектов другого класса (случайно попасть не в свою группу, а в другую). Как их разделять теперь, непонятно. И в этом случае у нас появляется отрицательный отступ, и все ломается.

Но есть выход. Можно просто ввести некоторый штраф за то, что объект попал к объектам другого класса.



Новая постановка задачи будет выглядеть так (в ней  $\xi_i$  как раз наш штрафной член):

$$\begin{cases} \frac{1}{2}\|w\|^2 + C \sum_{i=1}^l \xi_i \rightarrow \min_{w, w_0, \xi} \\ M_i(w, w_0) \geq 1 - \xi_i \quad i = 1, \dots, l \\ \xi_i \geq 0 \quad i = 1, \dots, l \end{cases}$$

Второе неравенство можем рассматривать как ограничение на  $\xi_i$ , если перенесем  $\xi_i$  влево, а *margin* вправо.

Тогда, чтобы  $\xi_i$  удовлетворяла обоим неравенствам (второму и третьему), нужно  $\xi_i \geq \max(1 - M_i(w, w_0), 0)$ .

$C$  – регуляризационная константа, которая отвечает за то, насколько сильно мы штрафуем модель.

В итоге сумму, которую мы минимизируем, можно переписать как

$$C \sum_{i=1}^l \max(0, 1 - M_i(w, w_0)) + \frac{1}{2} \|w\|^2 \rightarrow \min_{w, w_0}$$

Правда, сейчас мы хоть и учли ограничения, но получили более сложную для решения задачу, ведь  $\max(0, 1 - M_i(w, w_0))$  не является гладкой функцией.

Сейчас придется вспомнить методы оптимизации и теорему Каруша–Куна–Такера. В общем случае минимизация с заданными условиями это:

$$\begin{cases} f(x) \rightarrow \min_x \\ g_i(x) \leq 0 & i = 1, \dots, m \\ h_j(x) = 0 & j = 1, \dots, k \end{cases}$$

Тогда у нас есть необходимые условия для локального минимума в общем случае:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial w} = 0, \mathcal{L} = f(x) + \sum_{i=1}^m \mu_i g_i(x) + \sum_{j=1}^k \lambda_j h_j(x) \\ g_i(x) \leq 0; h_j(x) = 0 \\ \mu_i(x) \geq 0 \\ \sum \mu_i g_i(x) = 0 \end{cases}$$

Можем расписать для нашего Лагранжиана:

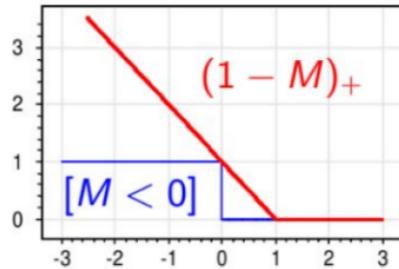
$$\mathcal{L}(w, w_0, \xi, \lambda, \eta) = \frac{1}{2} \|w\|^2 - \sum_{i=1}^l \lambda_i (M_i(w, w_0) - 1) - \sum_{i=1}^l \xi_i (\lambda_i + \eta_i - C)$$

Необходимые условия минимума:

$$\begin{cases} \frac{\partial \mathcal{L}}{\partial w} = 0 \Rightarrow w = \sum_{i=1}^l \lambda_i y_i x_i \\ \frac{\partial \mathcal{L}}{\partial w_0} \Rightarrow \sum_{i=1}^l \lambda_i y_i = 0 \\ \frac{\partial \mathcal{L}}{\partial \xi} = 0 \Rightarrow \eta_i + \lambda_i = C & i = 1, \dots, l \\ \xi_i \geq 0, \lambda_i \geq 0, \eta_i \geq 0 \\ \lambda_i = 0 \vee M_i(w, w_0) = 1 - \xi_i \\ \eta_i = 0 \vee \xi_i = 0 \end{cases}$$

Так причём здесь метод опорных векторов (SVM)? Делаем простое решение, в котором у нас участвуют только те объекты, которые так или иначе являются опорными объектами, то есть объектами, которые находятся около разделяющей полосы или внутри неё. Те объекты, которые лежат внутри своего класса, вообще говоря для нашего классификатора особо не играют роли, потому что та функция потерь, о которой мы говорим  $\max(0, 1 - M)$  реагирует только на те объекты, которые находятся внутри разделяющей полосы или в глубине чужого класса.

Наша модель штрафуется не только в том случае, когда она неправильно проклассифицировала объект, но и за то, что они просто попали внутрь разделяющей полосы.



На картинке наш настоящий лосс указан синей ступенькой, это тот лосс, который мы хотим минимизировать, а, значит, по рисунку видно, что тот функционал, который мы будем в итоге минимизировать, является верхней оценкой функции эмпирического риска.

По сути, минимизируя красный функционал (Hinge loss), мы минимизируем mis loss.

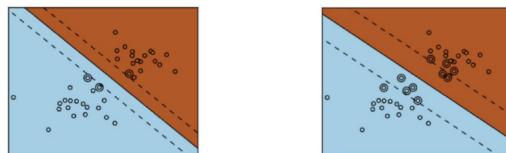
### Теперь про регуляризацию

Член функционала, состоящий из квадрата нормы вектора весов с некой константой - типичный регуляризационный член, от которого зависит ширина полосы. Напомним, что изначально мы хотели получить максимальную ширину между классами. Мы пришли к привычному виду функции потерь:

$$Q(w, w_0) = \sum_{i=1}^l [M_i(w, w_0) < 0] \leq \sum_{i=1}^l \max(0, 1 - M_i(w, w_0)) + \frac{1}{2C} \|w\|^2 \rightarrow \min$$

Первое слагаемое отвечает за аппроксимацию решения, а второе - за регуляризацию. Константа  $C$  в знаменателе регуляризационного члена появилась, потому что можно без нарушения общности поделить наш функционал на  $C$  (заметим, что это та же  $C$ , что и в формулах ранее).

Похожесть получившийся функции потерь на функцию потерь, например, в линейной регрессии, говорит нам о том, что есть общие подходы в математике построения моделей. И, вообще говоря, добавление регуляризации привносит добавление некоего априорного знания внутрь нашей модели, в данном случае - максимизирует ширину межклассовой полосы. Так как константа стоит тут в знаменателе, то чем больше константа, тем слабее регуляризация и тем полоса уже.



На этой картинке слева значение  $C$  больше, справа – меньше.  
**Нелинейный SVM**

Внутри *margin* лежит скалярное произведение. И, вообще говоря, мы можем использовать другое скалярное произведение, заданное для другого пространства.

Это основная идея нелинейного SVM: использовать другое преобразование для задания скалярного произведения на другом Гильбертовом пространстве. Для этого вводится функция ядра (*kernel*), которая обладает простыми свойствами.

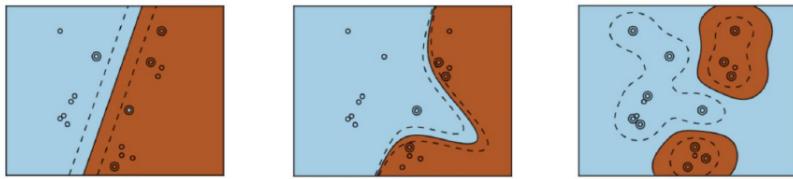
**Теорема 11.1** (теорема 1999 года Мерсера). Функция может являться ядром, если она симметрична и неотрицательно определена,

$$K(x, x') : X \times X \rightarrow \mathbb{R}$$

$$\exists \varphi : X \rightarrow H : K(x, x') = \langle \varphi(x), \varphi(x') \rangle, \text{ где } H \text{ — шильбертово пр-во}$$

**Пример 11.2.** Приведём несколько примеров ядер:

1.  $K(x, x') = \langle x, x' \rangle^2$
2.  $K(x, x') = \langle x, x' \rangle^d$
3.  $K(x, x') = (\langle x, x' \rangle + 1)^d$
4.  $K(x, x') = \exp(-\gamma \|x - x'\|^2)$  — RBF



На картинке слева разделяющая поверхность для скалярного произведения, по центру для полиномиального ядра, справа для *RBF*.

С различными ядрами мы можем строить различные разделяющие поверхности. Но проблема всё равно остаётся, популярность SVM упала за последние годы, так как чтобы подобрать ядро в таком алгоритме нужно перебрать возможные ядра, а возможные ядра подобрать «экспертным мнением». Тем более если у нас много задач, в каждой из которых своя выборка, то каждый раз придётся проделывать одну и ту же работу.

## 12 Principal component analysis. Relations to SVD. Eckart-Young theorem. How to apply PCA in practice.

Метод главных компонент во всех библиотечных реализациях называется PCA, существует для понижения размерности.

### Мотивация

Вспомним зачем вообще интересно понижать размерность.

1. В задачах ML размерность данных может быть очень большой. Например, миллионы признаков, особенно когда это разреженные признаки - становится сложно работать.
2. Такие данные сложно визуализировать, сложно обучать модели.
3. Чтобы снизить размерность, нам требуется использовать какой-то механизм, который по хорошему все-таки воспроизведим.

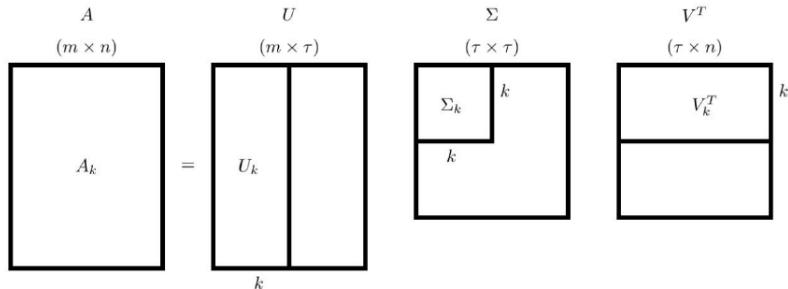
PCA - линейный способ понижения размерности, который интуитивен и используется широко на практике. Зачем нам может понадобиться матричное разложение на практике? Матрица объект-признак  $X$  может быть приближена произведением двух матриц меньшего ранга:

$$X_{l,d} \approx U_{l,k} V_{k,d}^T$$

Где размерность  $k < d, l$ . В таком случае мы можем сказать, что у нас есть матрица перехода и матрица, которая кодирует всю информацию. И теперь использовать приближение меньшего ранга для всех наших данных, потеряв часть информации, но используя меньше признаков.

Такое разложение можно построить с помощью минимизации  $\|X - UV^T\| \rightarrow \min$ . (Тут норма или норма Фробениуса, или какая-нибудь еще). PCA нам говорит, что мы можем матрицу признаков разложить на произведение трех матриц:  $X = U\Sigma V^T$ .

В этом выражении матрицы  $U, V^T$  являются ортогональными матрицами, а  $\Sigma$  - диагональная матрица. Можем сказать, что  $k$  ненулевых признаков остаются ненулевыми и значимыми. Получаем это с помощью взятия  $k$  диагональных элементов матрицы  $X$  и соответствующих им столбцов в матрицах  $U, V$ . Таким образом при перемножении получается итоговая матрица меньшей размерности, итоговый ранг равен  $k$ .



**Теорема 12.1** (Эскарта-Янга). SVD дает наилучшую аппроксимацию нашей изначальной матрицы

$$\begin{aligned} X_k &= U_k \Sigma_k V_k^T \\ \forall B_k : \text{rk}(B_k) &= k \\ \|X - B_k\|_F &\geq \|X - X_k\|_F \end{aligned}$$

Где  $B_k$  – матрица ранга  $k$ , полученная не по SVD разложению.

*Плюсы:*

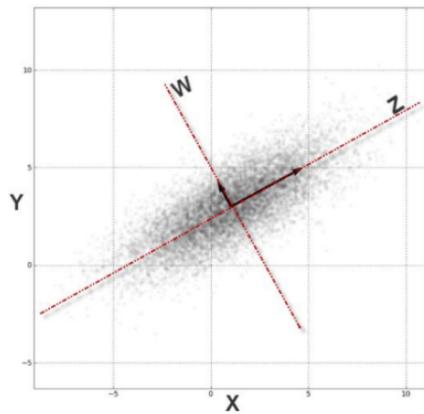
1. Так как используемые отображения линейны, значит и PCA - линейная модель, а значит будет работать достаточно быстро

2. преобразование, из которого можно будет вернуться в исходное пространство, пусть и с какими-то потерями (перешли в другое пространство с помощью  $U_k \Sigma_k$ , вернулись с помощью  $V_k^T$ ).

Так как у нас матрицы  $U, V$  ортогональны, давайте перетасуем все величины на диагонали матрицы  $\Sigma$  таким образом, чтобы они шли по убыванию. Первые  $k$  элементов – самые большие величины, которые позволяют описать данные наилучшим способом в  $k$ -мерном пространстве с точки зрения нормы Фробениуса.

## Интуитивная интерпретация РСА

Представим себе облако точек.



По сути мы переходим в базис главных компонент, которые можно заметить интуитивно: первая главная компонента – вдоль которой дисперсия максимальна (большая ось эллипса, определяем с точностью до направления), вторая и последующие главные компоненты будут ортогональны первой главной компоненте (так как мы помним, что компоненты матрицы ортогональны).

Мы работали на картинке в двумерном пространстве, получили две главные компоненты, теперь если мы захотим снизить размерность пространства, нам надо будет выбрать первую главную компоненту, чтобы не потерять максимальное количество информации.

**Замечание 12.2.** Вопрос, который любят задавать на собеседованиях: Каждая новая компонента смотрит в направлении максимальной дисперсии в оставшемся подпространстве, в котором мы теперь работаем. Все векторы ортогональны друг другу. Когда РСА выдаёт нам с ошибкой главные компоненты?

Понятно, что с точностью до направления они и так неоднозначно могут быть определены, но если у нас есть симметричность в данных – например, они лежат на поверхности сферы, тогда главные компоненты могут быть любые. То есть мы берем любые  $k$  ортогональных векторов, где  $k$  – размерность пространства.

**Пример 12.3** (Проблема). Как правило данные к нам приходят вообще из разных источников. Например, в выборке могут быть признаки: рост, вес, зарплата в долларах, зарплата в рублях и ещё 50 разных признаков. И мы можем обнаружить, что различные величины находятся в различных шкалах. Допустим рост в см от 50 до 200, вес от 40 до 140, а зарплата от 10 до 10000 в долларах и тд. Тогда получается, что чем больше разброс у того или иного признака, тем больше вдоль этого направления будет дисперсия, тем больше он на себя будет оттягивать внимания у

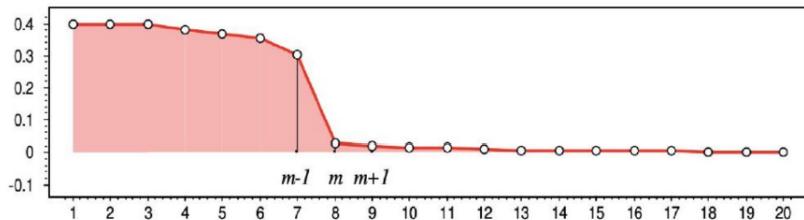
каждой следующей главной компоненты. Это неправильно, так как вместо того, чтобы смотреть на линейные подпространства и искать информативные комбинации исходных признаков, мы будем смотреть на признаки с большей дисперсией в используемых шкалах.

**Замечание 12.4** (Важно). В принципе проблема та же, с которой мы боролись  $L_2$ -регуляризацией. Чем больше разброс какого-то признака, тем меньше нам нужны веса для того чтобы моделировать поведение нашей модели. Поэтому если не нормировать данные перед PCA, PCA не даст адекватный результат.

Повторим, что шаг  $X_k = U_k \Sigma_k$  даёт понижение размерности, а шаг  $\bar{X} = X_k V_k^T$  возвращает нас в исходное пространство.

В принципе, нам может быть интересно посмотреть на отклонение  $\bar{X}$  от  $X$ .

**Пример 12.5.** Рассмотрим:



В какой-то момент количество дисперсий значительно падает, это значит, что мы нашли линейное подпространство, которое описывает 99% наших данных. И, на самом деле, это как раз то количество главных компонент, которые нужно использовать.

Собственно, на практике мы сначала строим PCA такой же размерности, как и наше пространство (или как максимальное число компонент, которые вообще чисто физически возможно взять). Дальше смотрим на график как из примера выше и уже берем только нужное нам количество.

## 13 Train, validation and test stages of model development. Overfitting problem, ways to detect it.

Для работы модели нужно определить ее гиперпараметры. Рассмотрим две идеи:

Первая идея (Train, validation and test stage, метод отложенной выборки): обучить модель по части train и по предсказаниям на второй части определить оптимальные гиперпараметры. Минус такого подхода в том, что модель будет зависеть от случайного выбора части train (а вдруг объекты в train упорядочены по алфавиту и не все объекты попадут в обучающую выборку при определении гиперпараметров. Модель с этими гиперпараметрами может показать плохой результат на всей обучающей выборке).

Вторая идея (кросс-валидация): часть объектов идет на валидацию, часть на train, затем данный процесс повторяется несколько раз (часть с валидацией меняется). Тогда мы можем измерить дисперсию ошибки для каждого набора гиперпараметров и сам лосс, чтобы выбрать оптимальный набор гиперпараметров.

**Подробнее про Train, validation and test stage:**

**Тренировочная выборка** - часть полной выборки, на котором мы обучаем модель.



Рис. 1: Train, validation and test.

*Валидационная выборка* - часть полной выборки, происходит настройка гиперпараметров. Это мы и назовём валидированием или валидацией модели. Изначально человек из каких-то знаний, опыта и других соображений выбирает некоторый пул гиперпараметров для текущей задачи. И далее по ним с каким-то шагом запускаем модель и проверяем получившиеся предсказания с помощью метрик качества модели. Выбирает и фиксирует значения гиперпараметров.

*Тестовая выборка* - часть полной выборки, который мы заранее отложили и проверяем значение функции потерь, а также метрики качества модели. НЕ НАДО ВАЛИДИРОВАТЬСЯ НА ТЕСТОВОЙ ВЫБОРКЕ. Считайте, что проверка на тестовой выборке - это как финальная проверка работы. Уже с оптимизированными параметрами и гиперпараметрами запускаем модель, чтобы проверить её качество на этой части данных. Иногда для конечного теста, в тестовую выборку докидывают и валидационный датасет, но такой шаг является спорным.

### Overfitting problem, ways to detect it.

**Определение.** Недообучение — ситуация, когда модель уловила не все общие закономерности и не способна достаточно точно воспроизвести распределение, из которого создаются объекты. 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.

**Определение.** Переобучение — ситуация, когда модель не только успешно смоделировала распределение, но и включила в него шумовые факторы (то есть переобучилась под выбросы). То есть построенная модель хорошо работает на объектах из тестовой выборки, но плохо работает на объектах, не участвовавших в тестовой выборке.

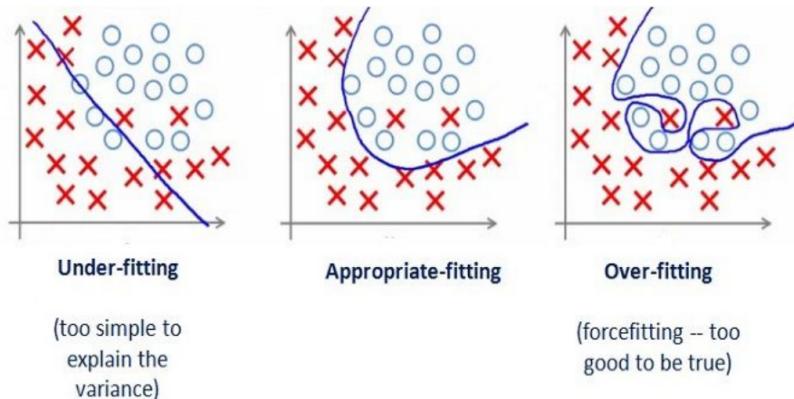


Рис. 2: Недообучение, хороший фит, переобучение.

### Как детектировать это все?

Вспомним про train и test, будем измерять лосс на них в зависимости от сложности модели (например, число эпох или норма вектора весов в линейной регрессии).

- Если и на train, и на test лосс падает, то мы уловили не все основополагающие зависимости  
⇒ недообучение.
- Если на train лосс падает, а на test лосс растет, то мы уловили шумовые зависимости, чуждые распределению ⇒ переобучение

Подробнее разберём вопрос: переобучение и недообучение. Когда мы просто запоминаем точки из тренировочной выборки, у нас будет не модель, которая будет возможно проходить через все точки на train, но на реальных данных показывать плохие результаты, модель скорее всего будет слишком сложной. А также если наша модель недообучилась, то будет слишком простая, неподходящая под реальность модель. Чем больше мы увеличиваем сложность моделей, тем больше данных нам нужно, чтобы её пофитить, но тем больше у нас возможностей изучить сложные закономерности. Чтобы баланс не нарушить, то есть иметь оптимальное обучение без недообучения и переобучения, нам нужно соблюдать все 3 этапа жизненного цикла построения модели (следует из bias variance decomposition). Оптимальной сложностью или временем обучения нейросети, является тот момент, когда значение функции потерь на валидационной выборке начинает расти при падении значения функции потерь на тренировочной выборке. Хорошо это понять можно по картинке ниже или при построении подобных графиков в реальной жизни - на следующем рисунке обучения бинарного классификатора.

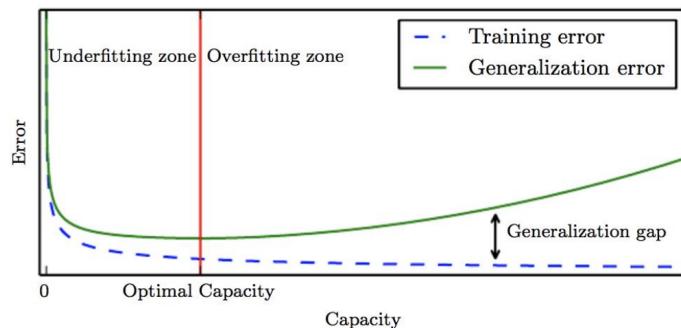


Рис. 3: Зелёной линией отмечено значение loss function на валидационной выборке, синей линией - на тренировочной. По оси x отмечена сложность модели (эпохи обучения). Красной линией отмечена оптимальная сложность: левее неё недообучение, правее - переобучение.

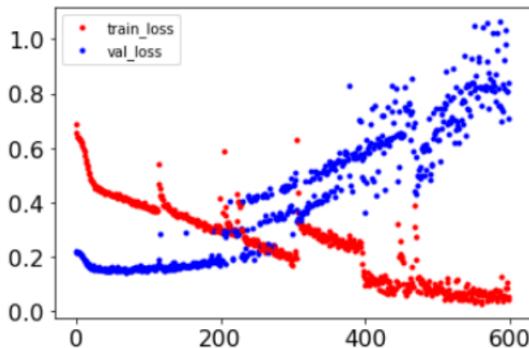


Рис. 4: Поведение функции потерь бинарного классификатора на валидационной и тренировочной выборке от эпохи обучения нейросети.

То есть переобучение - это когда значение функции потерь на валидационной выборке растёт, а на тренировочной падает. Недообучение - когда они все вместе убывают и ещё не надо останавливаться в обучении.

## 14 Validation strategies. Cross validation. Data leaks.

Вернёмся к идеи кросс-валидации:

**Кросс-валидация** - это когда мы делим выборку на  $k$  кусочков, на  $k-1$  используем как train, 1ый кусочек используем как валидацию. Потом берём второй кусочек, на остальных обучаемся

и предсказываем на 2ом. И так проходимся по всем кусочкам, то есть потребуется построить к моделям, тогда появится предсказание для каждого значения. В результате получается оценка эффективности выбранной модели с наиболее равномерным использованием имеющихся данных.

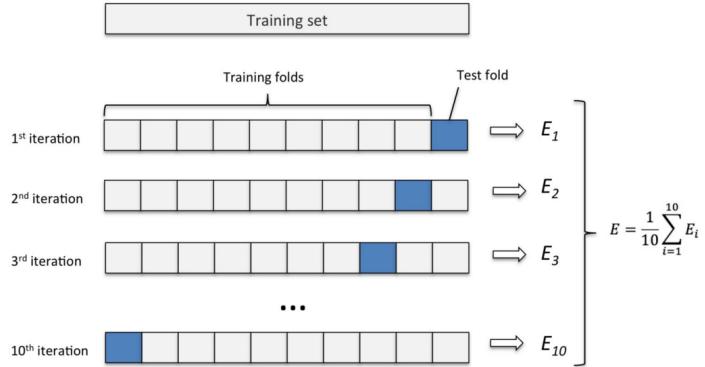


Рис. 5:

Не путать кросс-валидацию и валидацию в целом.

Как соотносятся размеры тренировочной и валидационной выборки? (Вопрос из зала)

От 90/10 до 70/30, зависит от задачи и количества данных.

Note. Далее речь идет о K-fold кросс-валидации, где К — число запусков кросс-валидации и одновременно то, на сколько примерно равных кусков мы бьем train.

Тогда в рамках каждого запуска один из К блоков является валидацией, а остальные — train.

Влияние количества блоков:

- При увеличении числа блоков дисперсия ответа уменьшается (закономерно, поскольку объем данных увеличивается)
- Аналогичная ситуация с bias - матожидание разности между истинным ответом и выданным алгоритмом - оно уменьшается.
- Чем меньше количество блоков, тем быстрее это работает, но тем меньше шанс, что модель подстроится под какой-то кусок данных.

Кросс-валидацию следует делать, когда данных мало, потому что она работает лучше. Но, когда выборки огромные, делать кросс-валидацию очень долго. Используем Train, validation and test stage поскольку количество данных и так позволяет хорошо обучить модель.

*Data leak* - утечка данных, при которой обучение происходит в том числе и на тестовых данных.

## 15 Bias-variance tradeoff.

**Прим.** Билет взят из открытых источников: [1](#) [2](#) (тема не найдена на лекциях)

Цель билета - рассказать о проблеме поиска баланса между недообучением и переобучением.

*Компромисс между смещением и дисперсией* - это свойство набора моделей предсказания, когда модели с меньшим отклонением (low bias) от имеющихся данных имеют более высокую дисперсию (high variance) на новых данных (то есть подвержены переобучению), и наоборот модели с меньшей дисперсией (low variance) от имеющихся данных имеют более высокое смещение (high bias) на новых данных (это приводит к недообучению). Ниже более подробное объяснение этого процесса. Возникает конфликт при попытке одновременно минимизировать эти два источника ошибок, которые мешают алгоритмам обучения с учителем делать обобщение за пределами тренировочного набора.

Смещение - это разница между средним прогнозом нашей модели и правильным значением, которое мы пытаемся предсказать. Модель с высоким смещением уделяет очень мало внимания

данным обучения и чрезмерно упрощает модель. Это всегда приводит к высокой погрешности в обучающих и тестовых данных. (недообучение).

Дисперсия - это изменчивость прогноза модели для данной точки данных или значения, которое говорит нам о разбросе наших данных. Модель с высокой дисперсией уделяет большое внимание обучающим данным и не обобщается на данные, которые она раньше не видела. В результате такие модели очень хорошо работают с обучающими данными, но имеют высокую частоту ошибок в тестовых данных. (переобучение)

### Математическое обоснование

Пусть переменная, которую мы пытаемся предсказать, равна  $Y$ , а другие ковариаты -  $X$ . Мы предполагаем, что между ними существует такая взаимосвязь, что  $Y = f(X) + e$ . Где  $e$  - член ошибки, и он обычно имеет матожидание 0.

Мы создадим модель  $\hat{f}(X)$ , используя линейную регрессию или любой другой метод моделирования. Таким образом, ожидаемая квадратическая ошибка в точке  $x$  равна

$$Err(x) = \mathbb{E}[(Y - \hat{f}(x))^2]$$

Можно разложить в:

$$Err(x) = (\mathbb{E}[\hat{f}(x)] - f(x))^2 + \mathbb{E}[(\hat{f}(x) - \mathbb{E}[\hat{f}(x)])^2] + \sigma_e^2$$

$$Err(x) = Bias^2 + Variance + IrreducibleError$$

$Err(x)$  - это сумма смещения<sup>2</sup>, дисперсии и неустранимой ошибки.

Неустранимая ошибка - это ошибка, которую нельзя уменьшить путем создания хороших моделей. Это показатель количества шума в наших данных.

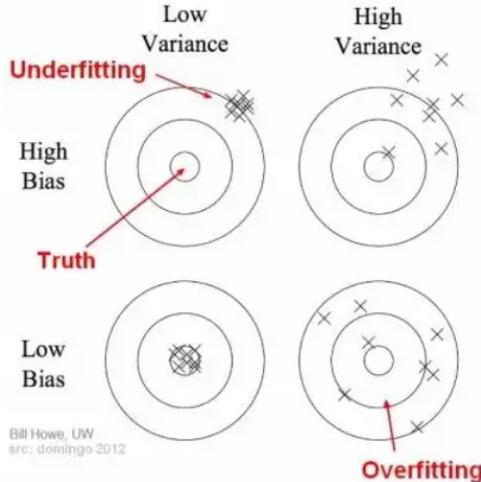


Рис. 6: Расположение тестовой выборки относительно модели.

### Проблема Bias-Variance Tradeoff

Это проблема сравнения  $Bias^2$  vs.  $Variance$ . Если наша модель слишком проста и имеет очень мало параметров, она всего будет иметь большое смещение и низкую дисперсию. С другой стороны, если наша модель имеет большое количество параметров, она будет иметь высокую дисперсию и низкое смещение. Поэтому нам нужно найти правильный/хороший баланс без переобучения и недообучения данных. Для этого, нам нужно найти хороший баланс между смещением и дисперсией таким образом, чтобы свести к минимуму общую ошибку.

$$TotalError = Bias^2 + Variance + IrreducibleError$$

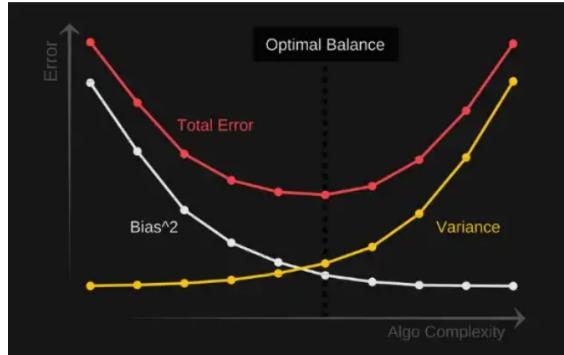


Рис. 7:

## 16 Decision tree construction procedure.

Решающие деревья.

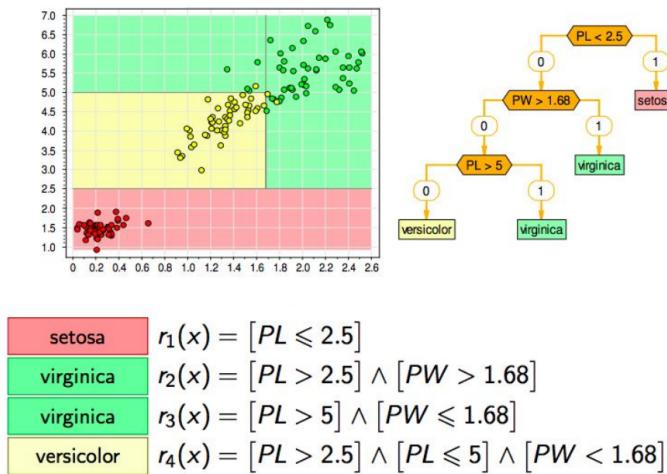


Рис. 8: Датасет Ирисы Фишера и классификатор в виде дерева.

Поговорим об интуиции, лежащей под капотом решающего дерева. Давайте взглянем на знакомый датасет Ирисы Фишера. На рисунке ниже можем увидеть алгоритм классификации в виде дерева, который на каждом шаге использует предикат и с помощью этого делает какое-то решение одно из двух. Делаем на первом шаге разделение на два листа по значению 1го признака. В первый класс попали все объекты класса setosa, в нулевой класс всё остальное. Теперь по значению второго признаку, мы делим оставшуюся часть пространства пополам. Всё, что попало в класс 1 - virginica, в класс 0 остались зелёные и жёлтые признаки. Следующий шаг опять в делении по уже другому трешхолду 1го признака на две выборки, в одной из них virginica, в другой желтый versicolor. На рисунке дерево глубины 3, но некоторые объекты проклассифицировались неправильно, поэтому мы можем достроить дерево более глубоким, чтобы все точки были проклассифицированы правильно. С одной стороны было бы хорошо так точно угадывать класс, но с другой стороны мы получим очень глубокое, то есть переобученное дерево. Так как мы пытаемся уловить возможно шумовые наблюдения в нашей обучающей выборке, мы переобучаемся под неё и теряем общую идею, которую мы могли бы перенести в тестовую выборку. Если присмотреться к такому классификатору, можно заметить, что каждому подпространству модель предсказывает константу. Если посмотреть на задачу регрессии на рисунке ниже, это видно ещё лучше. Тут изображён синус, который зашумлен и мы пытаемся описать эту гладкую функцию с помощью решающего дерева. В данном случае признаковое пространство 1мерное. Получается кусочно-постоянная функция. Потому что мы делим

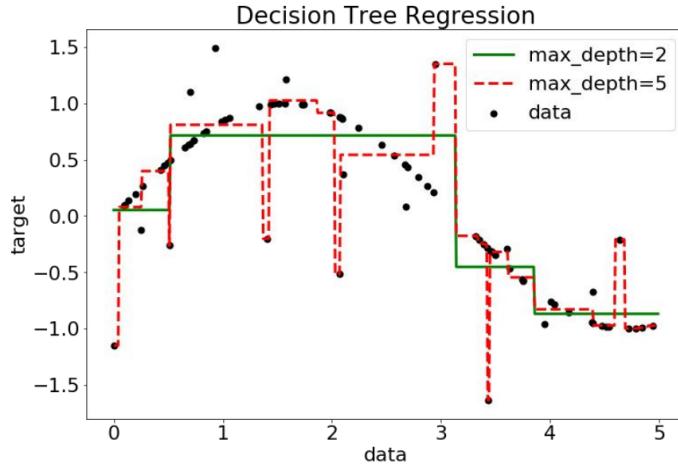


Рис. 9: Решающие деревья в задаче регрессии.

признаковое пространство на куски и каждому куску сопоставляем некоторую константу. Красным показано дерево глубины 5, а зелёным - глубины 2. Мы видим, что красное дерево поймало шумовые точки - переобучилось. Так получилось, потому что на этих точках была большая ошибка, дерево пыталось минимизировать ошибку и добилось такого фита. Дерево глубины 2 не очень хорошо описывает синус, но оно не переобучилось. Как нам построить дерево, а не просто применить построенное? Для этого нам требуется поставить оптимизационную задачу и обсудить как можно работать с деревом.

### 5.1 Построение модели решающего дерева

Сразу можно заметить, что поскольку функция, которую строит дерево кусочно-постоянная, а значит градиентные методы работать не будут (в каждом месте, где происходит разрыв, производная вообще не будет определена). Нам придётся вернуться к жадной оптимизации за неимением лучшего метода.

1. Пусть  $x^{(j)}$  -  $j$ -ый признак

$$x^{(j)} < t$$

В данный момент времени мы находимся в node дерева, и в ней мы делим все объекты по  $j$ -ому признаку, сравнивая его с трешхолдом  $t$ . Если признак у объекта меньше  $t$ , то объект идёт в левое поддерево, если больше - в правое поддерево. Получаются две новые ноды (это левое и правое поддерево).

2. Делаем тоже самое для каждой новой ноды. Рекурсивный алгоритм.

Откуда брать трешхолд и порог по индексу?

Поскольку датасет имеет конечное число признаков, которыми он описывается, то можно перебрать только конечное число трешхолдов, так как объектов тоже ограниченное количество. То есть будет 2 вложенных цикла: перебираем все возможные признаки, и все возможные трешхолды.

Нам нужен формальный критерий, по которому мы можем понять насколько качественное разбиение.

Пусть текущее условие  $x^{(j)} < t$  действует в области  $Q$ . Условие делит пространство  $Q$  на две части  $L$  и  $R$ . Введём функцию  $G$  - взвешенную сумму функции информативности от левой, правой подвыборок:

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

Логично, что в сумма  $\frac{L}{Q} + \frac{R}{Q} = 1$ .  $G(j, t)$  - указывает на то, насколько разбиение по  $t$  является хорошим для нас. Чем значение больше, тем лучше.

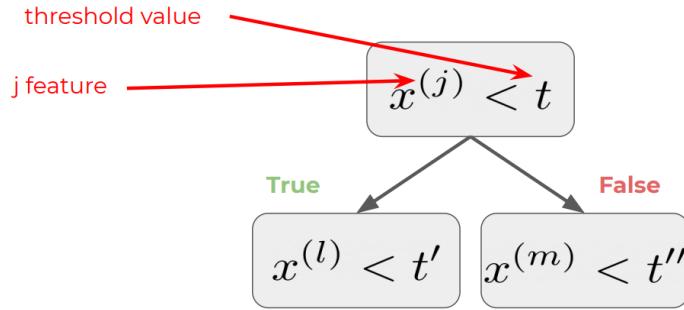


Рис. 10: Иллюстрация разбиения в узле.

Будем помнить, что дерево даёт только константу для каждого из листов (Это области, которые алгоритм не стал делить). Посмотрим на рисунок ниже. Для этого надо посчитать выборочные вероятности для классов встречающихся объектов в листе и выбрать наибольшую сумму.

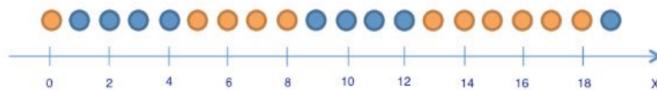


Рис. 11: Классификация по 1 признаку.

Есть задание простое задание классифицировать шарики. Признак - одно число. И два класса.

Если нам надо разделить на 2 признака выборку по какой-то границе, то скорее всего мы обрежем так, чтобы с одной стороны доминировал один класс, а с другой - другой.

Обратите внимание на картинку ниже. Введём понятие упорядоченности или информационного критерия  $H(R)$ . Под упорядоченностью понимается то, насколько сложно будет описать выборку, которая попала в лист. В простейшем случае бинарной классификации мы имеем дело с misclassification criteria:  $H(R) = 1 - \max\{p_0, p_1\}$ , где  $p_0$  - доля значений класса 0 в выборке  $R$ ,  $p_1$  - доля класса 1 в выборке  $R$ .  $p_0 + p_1 = 1$ .

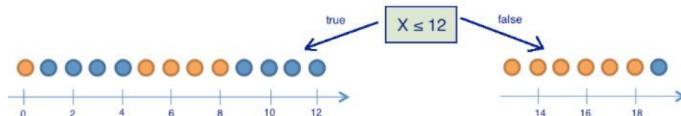


Рис. 12: Разбиение на 2 подвыборки.

Оценивает 1 - выборочную вероятность данного класса. То есть по сути это вероятность ошибиться, если мы предсказываем самый доминирующий класс. Не очень хороший подход, потому что он игнорирует все недоминирующие классы, для многоклассовой классификации это плохо. Такой критерий приведен скорее для исторической справки. Нам важны следующие 2 критерия:

1. Entropy criteria:

$$H(R) = -p_0 \log p_0 - p_1 \log p_1$$

2. Gini impurity:

$$H(R) = 1 - p_0^2 - p_1^2 = 1 - 2p_0p_1$$

Деревья, как и Наивный Байес работают с классификацией одинаково, вне зависимости от количества классов.

## 17 Information criteria. Entropy criteria, Giny impurity.

Хотим построить лист  $x^{(j)} < t$ . Хотим ввести какую-то характеристику «гетерогенности»  $H$  и минимизировать выражение

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

где  $Q$  - множество, которое разбивается этим листом, а  $L, R$  - части, на которое оно разбивается

Более простым языком:  $H$  - показывает насколько сильный разброс данных в этом множестве (чем он меньше, тем ближе все данные будут к предсказанному значению).

**Определение 17.1.** Функция  $H$  называется *критерием информативности* (information criteria)

Есть разные способы задать  $H$  для задачи классификации. Посмотрим сначала на примере бинарной классификации. Далее  $p_i$  - нормированная частота встречаемости  $i$ -ого класса в множестве  $R$

1. Misclassification criteria: количество ошибок классификации (предсказываем доминирующий класс и смотрим сколько объектов классифицировали неправильно)

$$H(R) = 1 - \max\{p_0, p_1\}$$

Не очень хороший способ: например для многоклассовой классификации дерево может хорошо отделять несколько каких-то классов от остальных, но этот критерий нам этого не покажет

2. Entropy criteria

$$H(R) = -p_0 \log p_0 - p_1 \log p_1 = -p_0 \log p_0 - (1-p_0) \log(1-p_0)$$

Энтропия «на пальцах»: чем больше энтропия, тем больше хаоса

3. Gini impurity

$$H(R) = 1 - p_0^2 - p_1^2 = 1 - p_0^2 - (1-p_0)^2 = 2p_0(1-p_0)$$

Интуитивно: берем два объекта из нашей подвыборки. Какова вероятность, что они окажутся разных классов?

Для многоклассовой классификации аналогично

1. Misclassification:  $H(R) = 1 - \max\{p_0, \dots, p_k\}$
2. Entropy:  $H(R) = -\sum_{i=0}^k p_i \log p_i$
3. Gini:  $H(R) = 1 - \sum_{i=0}^k p_i^2$

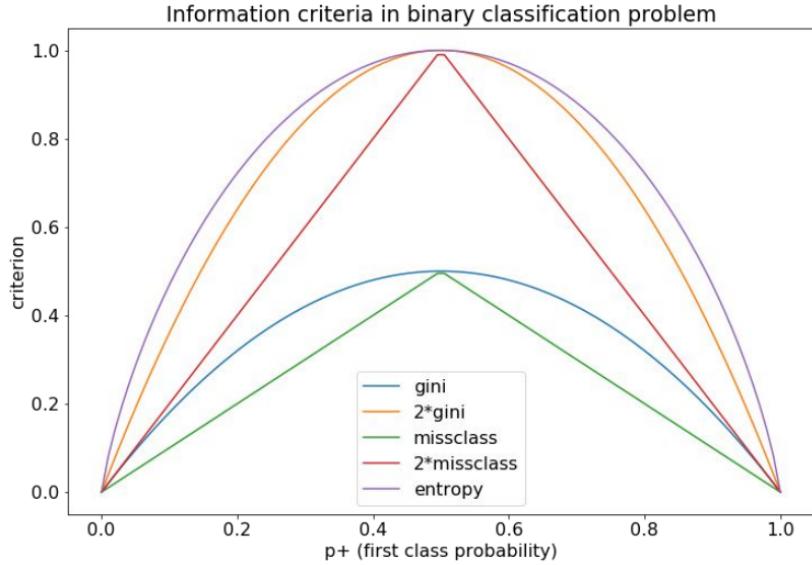


Рис. 13: График разных критериев информативности

Из графика выше видим, что энтропия и Джини ведут себя примерно одинаково и на практике не очень отличаются (вроде в последнее время чаще используют Джини). Так же график показывает, что misclassification слишком мало штрафует за малые ошибки (по сравнению с остальными критериями)

Для задачи регрессии нам придется взять другой критерий информативности (mean squared error)

$$H(R) = \min_c \frac{1}{|R|} \sum_{(x_i, y_i) \in R} (y_i - c)^2$$

Оптимальная константа (оценка максимального правдоподобия на  $c$ ):

$$c^* = \frac{1}{|R|} \sum_{y_i \in R} y_i$$

Интуитивно: считаем дисперсию относительно среднего

Вопрос «со звёздочкой»: если оптимизируем МАЕ, то  $c^*$  — это медиана

Подробнее про то, почему получаются такие оценки можно почитать [здесь](#)

## 18 Ensembling methods. Bootstrap. Bagging.

Рассмотрим выборку  $X$  размера  $m$ . Выберем  $m$  объектов с повторениями из  $X$  и получим *бутстррапированную выборку*. Повторим данный процесс  $N$  раз чтобы сгенерировать  $N$  выборок  $X_j$ .

Мы сделали это для того, чтобы получить много выборок похожих на  $X$ , но при этом не совпадающих с ним.

Возьмем нашу модель. Обучим ее на каждом  $X_j$  по отдельности. Введем обозначения

$$\varepsilon_j(x) = b_j(x) - y(x)$$

$$\mathbb{E}(b_j(x) - y(x))^2 = \mathbb{E}_x \varepsilon_j^2(x)$$

$$\mathbb{E}_1 = \frac{1}{N} \sum_{j=1}^N \mathbb{E}_x \varepsilon_j^2(x),$$

где  $b_j(x)$  - предсказание модели, обученной на выборке  $X_j$ ,  $y(x)$  - истинный ответ для объекта  $x$ ,  $\varepsilon_j(x)$  - ошибка  $j$ -ой модели,  $\mathbb{E}_1$  - средняя ошибка всех моделей

Предположим, что ошибки моделей обученных на разных бутстрэпированных выборках несмещены и нескоррелированы, т.е.

$$\mathbb{E}_x \varepsilon_j(x) = 0$$

$$\mathbb{E}_x \varepsilon_i(x) \varepsilon_j(x) = 0, \quad i \neq j$$

Рассмотрим модель, которая усредняет предсказания всех моделей полученных ранее. Обозначим ее предсказание за  $a(x)$ . Тогда

$$a(x) = \frac{1}{N} \sum_{j=1}^N b_j(x)$$

Найдем ошибку этой модели  $\mathbb{E}_N$

$$\begin{aligned} \mathbb{E}_N = \mathbb{E}_x (a(x) - y(x))^2 &= \mathbb{E}_x \left( \frac{1}{N} \sum_{j=1}^N b_j(x) - y(x) \right)^2 = \mathbb{E}_x \left( \frac{1}{N} \sum_{j=1}^N \varepsilon_j(x) \right)^2 = \\ &= \frac{1}{N^2} \mathbb{E}_x \left( \sum_{j=1}^N \varepsilon_j^2(x) + \underbrace{\sum_{i \neq j} \varepsilon_i(x) \varepsilon_j(x)}_{=0} \right) = \frac{1}{N} E_1 \end{aligned}$$

Получили, что ошибка упала в  $N$  раз. Что мы сделали не так? (если все хорошо, то изобрали какой-то гениальный метод и вся остальная машинка не нужна)

Ответ: предположение о несмещенности и нескоррелированности не всегда верно на практике.

Попробуем это исправить: с ошибкой каждой конкретной модели мы мало что можем сделать, но можем попробовать сделать так, чтобы ошибки разных моделей были не похожи друг на друга. Это должно уменьшить корреляцию ошибок и соответственно приблизить сумму из равенства выше к 0 и приблизиться к теоретическому результату. Воспользуемся слабостью деревьев: их склонностью к переобучению. Переобучим каждую модель под конкретную бутстрэпную выборку, и потом усредним результаты.

Но бутстрэпные выборки все же очень похожи, поэтому это не приведет нас к желаемому результату. Поэтому мы в каждой выборке хотим выбрать признаки, на которых будет обучаться модель. Подробнее об этом поговорим в следующем билете

**Определение 18.1.** Прием усреднения предсказаний модели, обученной на разных бутстрэпных выборках называется *бэггингом* (bagging = bootstrap aggregation)

**Замечание 18.2.** Усреднение по моделям работает только в том случае, если предсказания моделей осмыслены и мы просто хотим уменьшить дисперсию (понятно, что если модели выдают что-то рандомное никакое усреднение нам не поможет)

## 19 Random Forest, Random subspace method.

**Random Subspace Method (RSM):** случайно выбираем некоторые наборы признаков, обучаем модель только на них и усредняем результаты. Из-за того, что наборы признаков разные, ошибки не очень похожи друг на друга, а значит мы приближаемся к теоретическому результату из прошлого билета

**Random Forest:** объединим bagging и RSM на деревьях, чтобы ошибки стали еще менее похожими

Плюсы random forest: нелинейная разделяющая поверхность, сложно переобучить (переобучение всех деревьев усредняются, проблемы могут быть только если возьмем огромное количество деревьев и наборы признаков будут повторяться), не ломается при наличии скоррелированных признаков, деревья хорошо работают с пропусками (если нет признака, который нужен для этого листа, просто считаем значения в каждой из веток и усредняем)

$$\hat{y} = \frac{|L|}{|Q|}\hat{y}_L + \frac{|R|}{|Q|}\hat{y}_R$$

Так же random forest позволяет использовать обучающую выборку для валидации: так как мы обучаем каждое дерево на бутстральной выборке, есть объекты, которые каждая из моделей не видела. Поэтому можно для каждого объекта взять только те модели, которые его не видели и усреднить предсказания по ним. Получим оценку сверху на ошибку (так как уменьшили ансамбль, следовательно ошибка выросла).

$$OOB = \sum_{i=1}^m L \left( y_i, \frac{1}{\sum_{n=1}^N I[x_i \notin X_n]} \sum_{n=1}^N I[x_i \notin X_n] b_n(x_i) \right) \quad (\text{out-of-bag error})$$

**Замечание 19.1.** Если данные упорядочены во времени надо быть очень осторожными с ОOB, потому что важно смотреть распределение «новое при условии старого», а не наоборот (иначе модель обучается чему-то не тому)

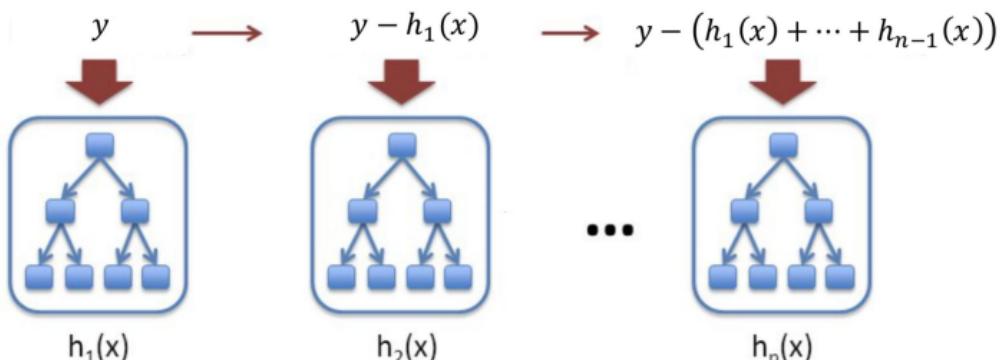
**Модификации:**

1. Extremely Randomized Trees: выбираем разделения в листах более рандомно (в экстремальном случае: деревья даже не зависят от ответов обучающей выборки)
2. Isolation forest (метод поиска аномалий): не предсказываем что-то, а пытаемся отделить аномалии, так как их проще отделить деревом чем остальные точки (этот метод не универсальный, работает не на всех задачах. Пример: точки на плоскости, «свернули в рулон» и добавили аномалии между слоями. Метод будет работать плохо)

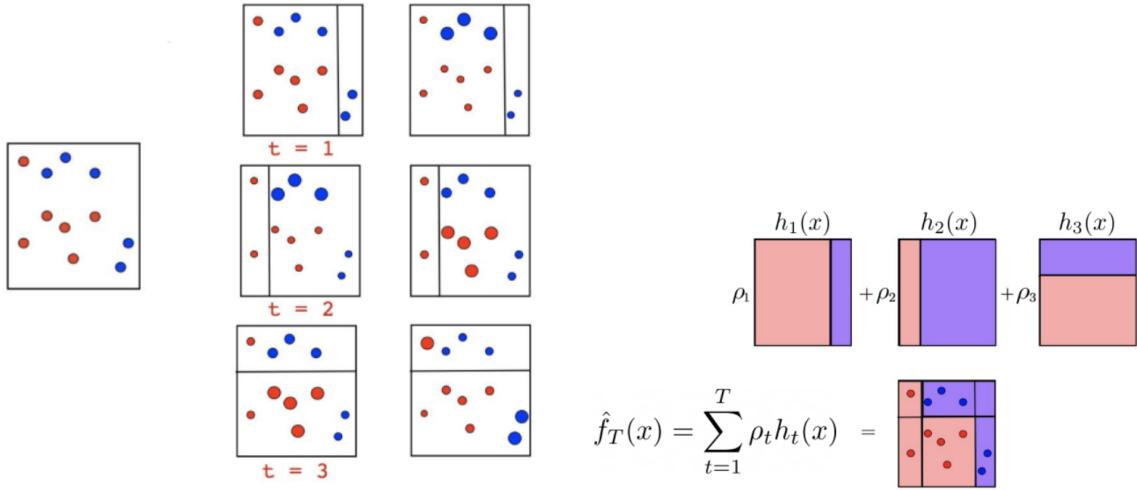
## 20 Boosting and gradient boosting. Main idea, gradient derivation.

Идея бустинга: берем несколько последовательных моделей. Каждая следующая предсказывает ошибку предыдущей

$$a_n(x) = h_1(x) + \dots + h_n(x)$$



**Пример 20.1.** Рассмотрим игрушечный пример на задаче бинарной классификации (см. картинки ниже). Каждая модель будет «решающим пнем» (деревом с одним разделением). После того как обучим все модели с учетом ошибок предыдущих (размер точек показывает ошибку) сложим их с какими-то весами и получим итоговую нелинейную модель



**Пример 20.2.** Рассмотрим экспоненциальную функцию потерь  $E(M) = e^{-M}$ . Тогда

$$\hat{f}_T = \sum_{t=1}^T \rho_t h_t(x)$$

$$\begin{aligned} L(y_i, \hat{f}_T(x_i)) &= \exp(-y_i \hat{f}_T(x_i)) = \exp(-y_i \sum_{t=1}^T \rho_t h_t(x_i)) = \underbrace{\exp\left(-y_i \sum_{t=1}^{T-1} \rho_t h_t(x_i)\right)}_{\text{const on step T}} \cdot \exp(-y_i \rho_T h_T(x_i)) = \\ &= w_i \cdot \exp(-y_i \rho_T h_T(x_i)) \end{aligned}$$

Видим, что у каждого объекта есть вес, зависящий от ошибок прошлых моделей, а на шаге  $T$  нам нужно минимизировать только последнюю функцию. Такой бустинг называется *AdaBoosting* (адаптивный). Однако он был не очень хорош: легко переобучался, экспонента мешала вычислительно и не всегда подходила под задачу

**Gradient boosting:** идея то же, что и в градиентном спуске, но в пространстве моделей

Рассмотрим датасет  $\{(x_i, y_i)\}_{i=1,\dots,n}, L(y, f)$  - функция потерь. Тогда оптимальная модель  $\hat{f}$  имеет вид

$$\hat{f}(x) = \operatorname{argmin}_{f(x)} L(y, f(x)) = \operatorname{argmin}_{f(x)} \mathbb{E}_{x,y} L(y, f(x))$$

Пусть модель взята из параметрического семейства

$$\hat{f}(x) = f(x, \hat{\theta})$$

$$\hat{\theta} = \operatorname{argmin}_{\theta} \mathbb{E}_{x,y} L(y, f(x, \theta))$$

Рассмотрим шаг индукции: пусть у нас обучены уже  $t - 1$  последовательных модели. Обозначения:

$$\hat{f}(x) = \sum_{i=0}^{t-1} \hat{f}_i(x)$$

$$(\rho_t, \theta_t) = \operatorname{argmin}_{\rho, \theta} \mathbb{E}_{x,y} L(y, \hat{f}(x) + \rho \cdot h(x, \theta))$$

$$\hat{f}_t(x) = \rho_t \cdot h(x, \theta_t)$$

Рассмотрим антиградиент функции ошибки по предсказанию

$$r_{it} = - \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x)=\hat{f}(x)}, \quad i = 1, \dots, n$$

Настроим нашу следующую модель на предсказание антиградиента

$$\theta_t = \operatorname{argmin}_{\theta} \sum_{i=1}^n (r_{it} - h(x_i, \theta))^2$$

Затем найдем оптимальный шаг в сторону градиента

$$\rho_t = \operatorname{argmin}_{\rho} \sum_{i=1}^n L(y_i, \hat{f}(x_i) + \rho \cdot h(x_i, \theta_t))$$

База индукции: так как каждая следующая модель исправляет ошибку предыдущей нам в принципе неважно с чего начинать, так что можно, например, начать с константы (среднее значение по всему датасету)

Частный случай для линейной регрессии и  $L = MSE$

$$r_{it} = - \left[ \frac{\partial L(y_i, f(x_i))}{\partial f(x_i)} \right]_{f(x)=\hat{f}(x)} = -2(\hat{y}_i - y_i)$$

Получили, что антиградиент пропорционален ошибке, то есть каждая следующая модель на прямую пытается предсказать ошибку предыдущих (в общем случае с другими моделями/функциями потерять это может быть не так)

**Замечание 20.3.** Важно следить за тем, чтобы градиентный бустинг не переобучился: если переобучится одна модель в цепочке, то все что дальше происходит уже бесполезно. Поэтому если вы пока не понимаете специфику задачи, лучше использовать неглубокие деревья в бустинге (так как их очень сложно переобучить)

**Замечание 20.4.** Сам градиентный бустинг не параллелизуется, но так как обычно его делают на деревьев можно распараллелить процесс построения каждого дерева

## 21 Matrix calculus and matrix derivatives. How to get the derivative of matrix/dot product

**Базовые операции:**

$$\begin{aligned} A(B+C) &= AB + AC, \\ (A+B)^T &= A^T + B^T, \\ (AB)^T &= B^T A^T, \\ (AB)^{-1} &= B^{-1} A^{-1}, \\ (A^{-1})^T &= (A^T)^{-1}. \end{aligned}$$

**Производные следа и определителя:**

$$\begin{aligned} \frac{\partial}{\partial A} \operatorname{tr} AB &= B^T, \\ \frac{\partial}{\partial A} \det A &= (\det A)(A^{-1})^T, \\ \frac{\partial}{\partial x} \log \det A(x) &= \operatorname{tr} \left( A^{-1} \frac{\partial A}{\partial x} \right). \end{aligned}$$

**След и определитель:**

$$\begin{aligned} \det(AB) &= \det A \det B, \\ \det(A^{-1}) &= 1/\det A, \\ \det A &= \prod_j \lambda_j, \\ \operatorname{tr} A &= \sum_j A_{jj} = \sum_j \lambda_j, \\ \operatorname{tr}(ABC) &= \operatorname{tr}(BCA) = \operatorname{tr}(CAB). \end{aligned}$$

**Производные:**

$$\begin{aligned} \frac{\partial}{\partial x} x^T a &= a, \\ \frac{\partial}{\partial x} x^T Ax &= (A + A^T)x, \\ \frac{\partial}{\partial x} x^T Ay &= xy^T, \\ \frac{\partial}{\partial x} A^{-1} &= -A^{-1} \frac{\partial A}{\partial x} A^{-1}. \end{aligned}$$

## Примеры

1.

$$y = x^T x, \quad x \in \mathbb{R}^N$$

$$\begin{aligned}\frac{d}{dx_i} x^T x &= \frac{d}{dx_i} \sum x_i^2 = \frac{d}{dx_i} x_i^2 = 2x_i \\ \frac{dy}{dx} &= 2x\end{aligned}$$

2.

$$\begin{aligned}tr(AB) &= \sum_{i=1}^n a_{1i}b_{i1} + \dots + \sum_{i=1}^n a_{ni}b_{in} \implies \frac{dy}{da_{ij}} = b_{ji} \\ \frac{dy}{dA} &= B^T\end{aligned}$$

3.

$$\begin{aligned}y &= a^T x \\ \frac{d}{dx_i} a^T x &= \frac{d}{dx_i} \sum a_i x_i = a_i \implies \frac{d}{dx} a^T x = a\end{aligned}$$

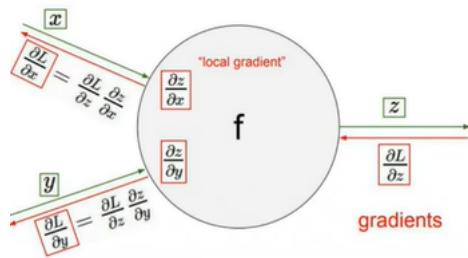
4.

$$\begin{aligned}y &= Ax \\ \frac{d}{dx_i} [Ax]_k &= \frac{d}{dx_i} \sum_j a_{kj} x_j = a_{kj} \\ \frac{d}{dx} Ax &= A\end{aligned}$$

## 22 Backpropagation, chain rule.

**Chain Rule:**

$$\frac{dL}{dx} = \frac{dL}{dz} \frac{dz}{dx}$$



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

На картинке выше показан шаг BackPropagation 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 входам, т.е.  $\frac{dz}{dx}$  и  $\frac{dz}{dy}$ , 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.

## Матричная форма

$$\begin{aligned}
y_1 &= f_1(\mathbf{x}) = x_1 \\
y_2 &= f_2(\mathbf{x}) = x_2 \\
&\vdots \\
y_n &= f_n(\mathbf{x}) = x_n
\end{aligned}$$

$$\frac{\partial \mathbf{y}}{\partial \mathbf{x}} = \begin{bmatrix} \nabla f_1(\mathbf{x}) \\ \nabla f_2(\mathbf{x}) \\ \vdots \\ \nabla f_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial \mathbf{x}} f_1(\mathbf{x}) \\ \frac{\partial}{\partial \mathbf{x}} f_2(\mathbf{x}) \\ \vdots \\ \frac{\partial}{\partial \mathbf{x}} f_m(\mathbf{x}) \end{bmatrix} = \begin{bmatrix} \frac{\partial}{\partial x_1} f_1(\mathbf{x}) & \frac{\partial}{\partial x_2} f_1(\mathbf{x}) & \dots & \frac{\partial}{\partial x_n} f_1(\mathbf{x}) \\ \frac{\partial}{\partial x_1} f_2(\mathbf{x}) & \frac{\partial}{\partial x_2} f_2(\mathbf{x}) & \dots & \frac{\partial}{\partial x_n} f_2(\mathbf{x}) \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial}{\partial x_1} f_m(\mathbf{x}) & \frac{\partial}{\partial x_2} f_m(\mathbf{x}) & \dots & \frac{\partial}{\partial x_n} f_m(\mathbf{x}) \end{bmatrix}$$

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.

## 23 Neural network concept. Fully-Connected layer (FC). Logistic regression as simple NN. XOR problem.

### 23 NN concept. Fully-connected layer.

#### Log reg as NN. XOR problem

- Появился > 70 лет назад
- 1943 Мат. модель нейрона. McCulloch & Pitt  
→ активация
- 1958 Персептрон Розенблата  
→ чем-то похожа на нейросети, умела разделять классы
- 1969 XOR проблема. Минский и Паперт  
Обычный персептрон (однослойные нейросети)  
не может решить проблему XOR ( $x_1 \oplus x_2$ )  
т.к. это сумма она линейна.  
(может только с дополнительной фишкой  $x_1 * x_2$ )

! 1-слойные НН: логистическая регрессия

Логистическая фун.:  $X \rightarrow Wx + b \rightarrow \sigma \rightarrow P(y)$   
нелин. предпр. сигмоид

Все дифференцируемо

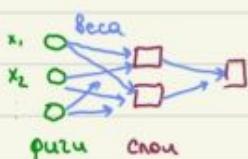
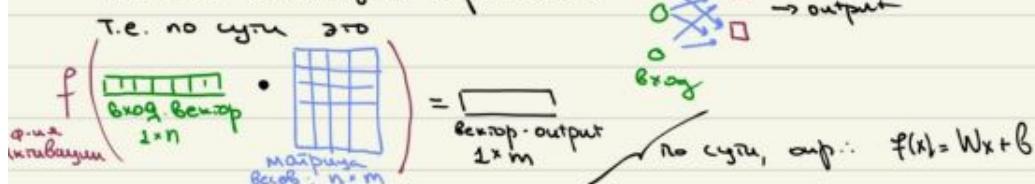
Classic pipeline  $X \rightarrow$  Feature Extractor  $\rightarrow$  Classifier  $\rightarrow$  Предсказание

NN pipeline  $X \rightarrow$  Linear model  $\rightarrow \sigma(W) \rightarrow$  Logistic Regression  $\rightarrow$  Предсказание

Fully-connected layer: каждый "нейрон" связан с

каждой входящей переменной

т.е. по сути это



• 1974 - Backpropagation

Хитом

• 1986 - Многослойный персептрон Красноярская группа

• 90-е Сверточные нейросети LeNet - 1998

AlexNet - 2012

• 2014 - GAN

• 2017 и далее - Трансформер, BERT

## 24 Losses for NNs: logistic loss, cross-entropy.

### 24 Losses for NN

не нашёл в лекциях

написан через конспект 2017 в интернете

#### ① Logistic loss

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

Используется для оценки вероятности

то же самое, что кросс-энтропия.

Используется в проблемах многоклассовой классификации

#### ② Hinge . аналогично

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

ког. во  
объектах

правильный  
ответ для

i-ого объекта

абс. отклонение о принадл.  
i-ого объекта к j-ому  
классу

$$\text{③ } l_2: \text{класса } l_2 = \frac{1}{n} \sum_{i=1}^n (\hat{y}_i - y_i)^2$$

Не исп. в задачах многоклассовой  
классификации.

## 25 Activation functions, their impact on the network, computational complexity. Softmax and LogSoftmax activations, numerical stability.

**25. Функции активации, их влияние.**

**Близость слоев Softmax, LogSoftmax, numerical stability**

Угол: глобальных минимумов в нашей нейроне

Изображено зона стабильной функции

- 0 - не активирующая, 1 - активирующая
- Норма дифф. фнк., т.ч. (но не вб-но) обр. фнк.  $\in [0, 1]$
- Opt. function, applied to layer output

**Сигмоиды - логарифмическая аппроксимация ненулевого значения**

$f(a) = \frac{1}{1 + e^{-a}}$  sigmoid

$f(a) = \tanh(a)$

$f(a) = \max(0, a)$  ReLU

$f(a) = \log(1 + e^a)$  softplus

**ReLU, max(0, x)**  
leaky ReLU, max(0.01x, x)

ELU:  $\begin{cases} x, x > 0 \\ 1(e^{-|x|}), x \leq 0 \end{cases}$

|                   | $x < 0$  | $x > 0$  |
|-------------------|--|--|
| ① Сигмоиды        | Гладкая аппроксимация ненулевого фнк.  | Гладкость близка к 0 (стабильна)                         |
| ② Тангенс         | Справка 0. В остальном похож на сигмоиду   | Не однородная для 0 и выше<br>Следует вычесть близко к 0 |
| ③ ReLU ✓<br>базис | Считается очень быстро, в $\approx 10^6$ раз быстрее сигмоиды<br>Но исключается область от 0 | Не однородна<br>График 0 при $x < 0$                     |
| ④ Leaky ReLU      | Не засыхает $\Rightarrow$ чуть лучше ReLU  |  |
| ⑤ ELU             | Красиво аппроксимирует $x < 0 \rightarrow$ const при $x < 0$                                 | Все выше надо считать $e^x$                              |

Softmax:  $G(z)_j = \frac{e^{z_j}}{\sum_{k=1}^n e^{z_k}}$  LogSoftmax:  $\log G(z)$

При больших  $x$  график у softmax может неконтролируемо уменьшаться

Numerical stability: softmax легко выходит NaN из-за overflow,

таким, что  $\text{softmax}(x) = \text{softmax}(x + c)$

$\Rightarrow$  будем считать  $\text{softmax}(x - \max(x))$

использование для вероятностей в задаче многоклассовой классификации.

## 26 Optimization methods in Deep Learning. Gradient descent, SGD, its upgrades: Momentum, RMSProp, Adam.

**26 Optimization methods in DL**

**Gradient Descent, SGD + upgrades**

Stochastic gradient descent is used to optimize NN parameters.

$$x_{t+1} = x_t - \underbrace{\text{learning rate}}_d \cdot \underbrace{\nabla f(x_t)}_{\text{gradient}}$$

GD: см. раньше, нужно считать градиент  $\nabla f(x)$  для оптимизировать loss

SGD: берёт несколько батчей, но них считает loss и градиент, усредняет (таким образом более точные шаги)

**Упражнение:**

- ① Momentum / метод тангенциального шага
- Аналогично градиенту:  $v_{t+1} = \gamma v_t + \nabla f(x_t)$
- с шагом:  $x_{t+1} = x_t - d v_{t+1}$
- Делает шаги на "половине" склонов
- метод называется "тангенциальный" learning rate:  $\gamma < 1$
- momentum
- actual step
- gradient step

② Hessian - аналогично 1, но считает ковариационную матрицу градиента

③ Adagrad: SGD with cache:  $Cache_{t+1} = Cache_t + (\nabla f(x_t))^2$

$$x_{t+1} = x_t - d \frac{\nabla f(x_t)}{Cache_{t+1} + \epsilon}$$

! Не работает для симметричных градиентов

④ RMSProp:  $Cache_{t+1} = \beta Cache_t + (1-\beta)(\nabla f(x_t))^2$

⑤ Adam: momentum + RMSProp

$$v_{t+1} = \gamma v_t + (1-\gamma)\nabla f(x_t) \quad Cache_t = \beta Cache_{t-1} + (1-\beta)(\nabla f(x_t))^2$$

$$x_{t+1} = x_t - d \frac{v_{t+1}}{Cache_{t+1} + \epsilon}$$

! Экспоненциальное сглаживание - Пуск с набором  $\{x_n\}$  в коэффиц.  $\alpha \in [0; 1]$ . Тогда  $S_n = \alpha S_{n-1} + (1-\alpha)x_n$  - эксп. сгл., т.к. предыдущее значение фиксируется в  $S_n$  с экспоненциальной убывающей скоростью

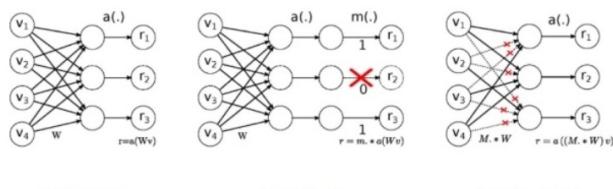
## 27 Regularization in Deep Learning: Dropout, Batch Normalization. Differences in training and evaluation stages.

### 27. Regularization in Deep Learning Dropout, BatchNorm. Diff in train & eval

→ плохо, потому что в итоге модель может выдавать это же то, (пример: стратифицирован в train-test split)

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 → если слишком много, то понижает эффективность (точность vs устойчивость)
- Dropconnect** - аналогично dropout, но действует не на выходы нейронов, а на ребра, то есть случайно обнуляет каждый переход (элемент матрицы переходов), обычно применяется в полносвязных 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)  $\star$  регуляризующие

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

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

Стратификация: идея: пусть у нас есть "красные" и "зелёные".

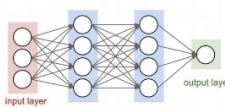
Тогда мы хотим, чтобы в train и test тоже было 10% красных и 90% зелёных, иначе есть шанс, что train будет количество состоять из "зелёных"

\* Такое это признак переобучения, т.к. у "близких" значений x разные значения y.

### BatchNorm - 2015 год

Problem:

- Consider a neuron in any layer beyond first
- At each iteration we tune it's weights towards better loss function
- But we also tune it's inputs. Some of them become larger, some - smaller
- Now the neuron needs to be re-tuned for it's new inputs



⇒ Идея: хорошо бы нормализовать нейтральное слое

↳ Часто надо позубить после ReLU с интуицией

того, что ReLU выдает на mean+var, а batch norm делает так, чтобы это не влияло на след. слой

! Хорошо описано в статье рекомендует использовать Batch Norm до ReLU, практика и здравый смысл показывают обратное

- Normalize activation of a hidden layer (zero mean unit variance)

$$h_i = \frac{h_i - \mu_i}{\sqrt{\sigma_i^2}}$$

- Update  $\mu_i, \sigma_i^2$  with moving average while training

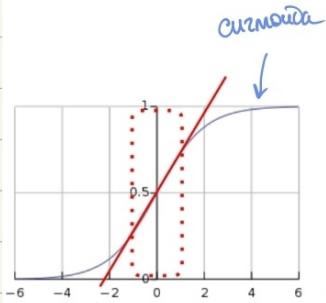
во время  
нормализации  
используют  
накопленные

$$\mu_i := \alpha \cdot \text{mean}_{\text{batch}} + (1 - \alpha) \cdot \mu_i$$

$$\sigma_i^2 := \alpha \cdot \text{variance}_{\text{batch}} + (1 - \alpha) \cdot \sigma_i^2$$

↙ оригиналные статьи

**Input:** Values of  $x$  over a mini-batch:  $\mathcal{B} = \{x_1 \dots m\}$ ;  
**Parameters to be learned:**  $\gamma, \beta$   
**Output:**  $\{y_i = BN_{\gamma, \beta}(x_i)\}$



сигмоида  $[-1, 1]$   
 норма линейная  
 $\Rightarrow$  BatchNorm<sup>+</sup> - это  
 сигмоида (но с гораздо  
 меньшей яркостью)  
 от ненормализованных  
 добавили

+ Data augmentation: при подготовке данных из  
 картинок увеличив его, скажем преобразовать:  
 обрезав/изменив яркость.

Условно, значение на картинке кота   
 не зависит от ее яркости.

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

внешние параметры, обратноющие  
 из  $[ ]$  в  $[\beta, \gamma + \beta]$   
 в случае, если модели  
 надо,  $\gamma = \text{дисперсия}$   $\beta = \text{среднее}$   
 $\Rightarrow$  модель имеет "отметки"  
 Batch Norm

## 28 Vanilla Recursive NN cell. Backpropagation through RNN. Vanishing gradient problem. Potential solutions.

### 28. Vanilla Recursive NN cell. Backpropagation through RNN Vanishing gradient problem + solutions

Оп.: ноды - дискретное значение (ДНК) - обычно обработка текстов  
временной ряд - непрерывное значение (т на узле)

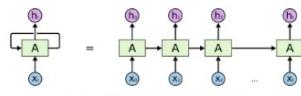
Токен - наименьший смысловой элемент ноды (буква, слово...)  
 ↳ можно видеть разные (зависит от задачи)

"Гуашь" варят работая с текстом: Bag of words

Разделяем на токены и считаем кол-во раз, сколько встретились

(каждый токен имеет векторизацию, например,  
 one hot encoding)

14.1 Рекуррентные НС



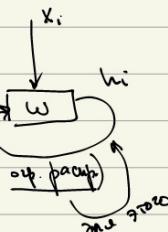
An unrolled recurrent neural network.  
 Рекуррентные нейронные сети состоят из последовательности скрытых состояний, в которые ведут входы  $x_i$  и выходы  $h_i$ . Таким образом они могут принимать вход последовательности из нескольких объектов. Например, тексты.

Формула для очередного состояния:

$$h_i = \sigma(W_{ih}h_{i-1} + W_{ix}x_i + b)$$

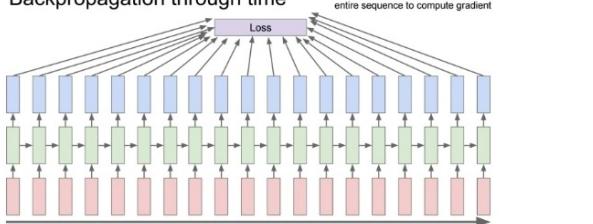
Таким образом, мы умножаем текущее состояние на матрицу, а затем прибавляем к ней текущий вектор умноженный на другую матрицу весов, накладываю на всё это величину.

$$h_i = G(W[h_{i-1}; x_i] + b)$$



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 всей последовательности. Мы «развертываем» рекуррентную сеть в обычную feed-forward, но чтобы посчитать производную на шаге backprop, мы должны сложить все производные и обновить матрицу  $W_{hh}$ , тут кроется проблема. Мы очень много раз применим коррелированные обновления, это очень плохо оказывается на SGD.

Появляется программа изымавших или затухающих градиентов.

$$h_0 = \bar{0}$$

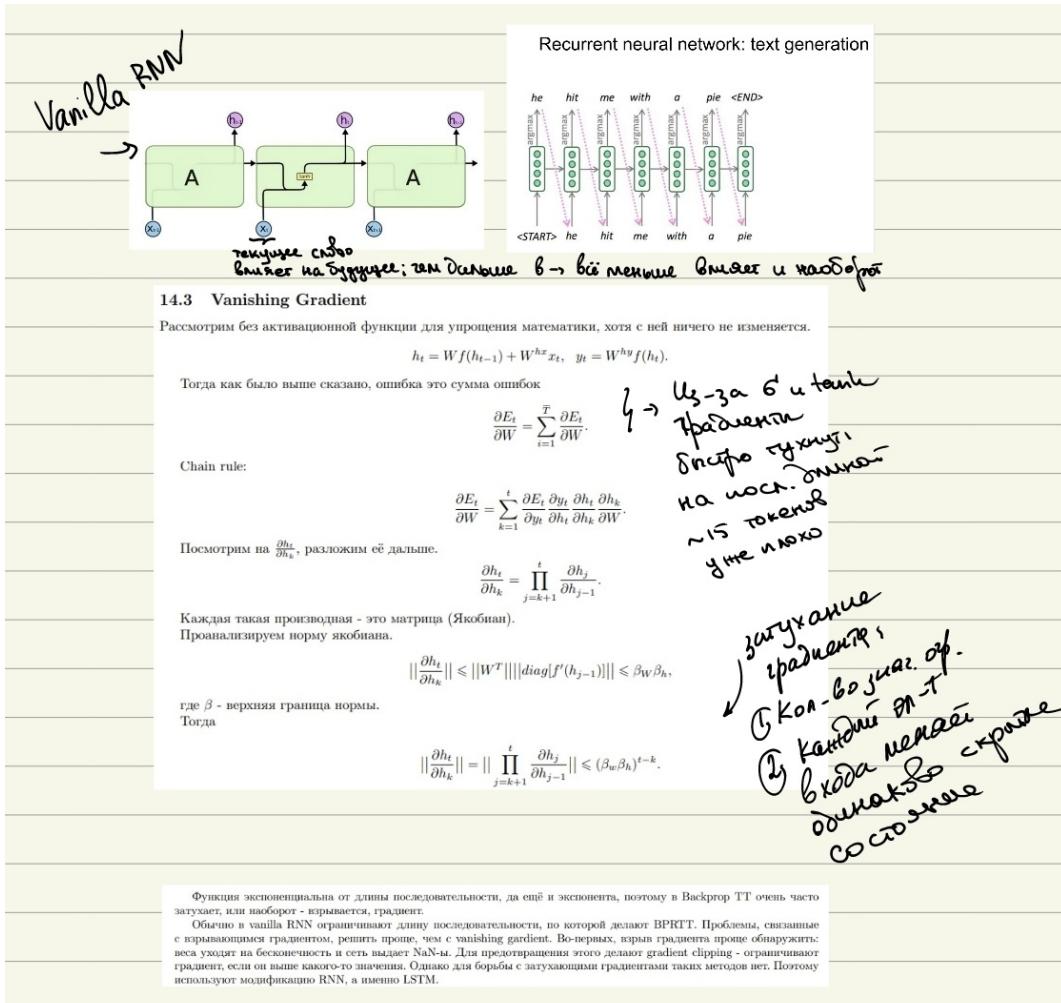
$$h_1 = \sigma(W_{hid}[h_0, x_0] + b)$$

$$h_2 = \sigma(W_{hid}[h_1, x_1] + b) = \sigma(W_{hid}[\sigma(W_{hid}[h_0, x_0] + b), x_1] + b)$$

$$h_{i+1} = \sigma(W_{hid}[h_i, x_i] + b)$$

$$P(x_{i+1}) = \text{softmax}(W_{out}, h_i) + b_{out})$$

50



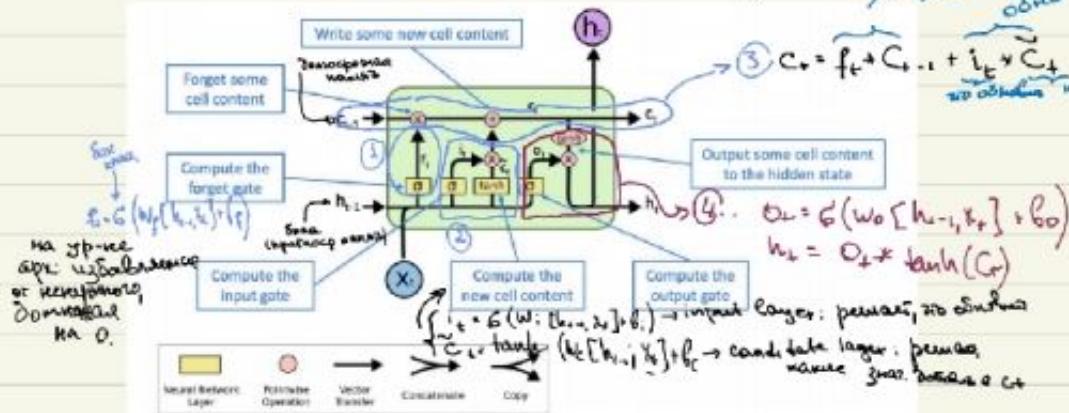
## 29 LSTM/GRU, memory concept, gates ideas.

### 29 LSTM/GRU, memory concept, gate ideas

LSTM: яким можимо, ненужні токи видає

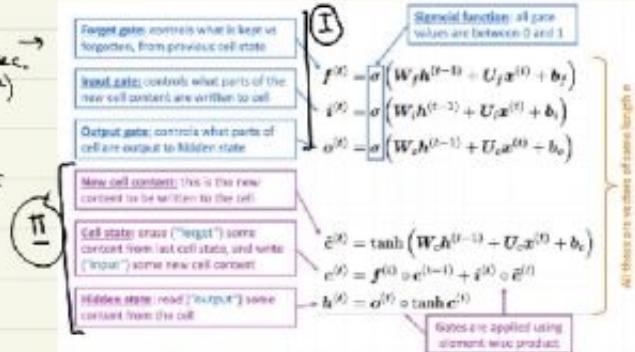
→ задаючи відмінність (forget gate)

забудований  
зас, змінний від  
одного до іншого.



### I) Якщо не зберігати (занадто пам'яті) II) Додавання пам'яті (занадто пам'яті)

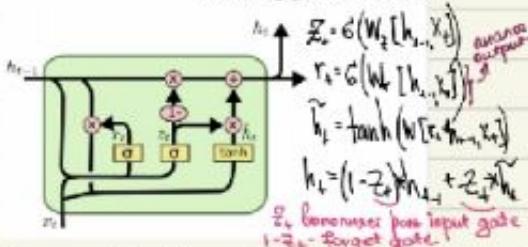
① Зберігати пам'яті  
безп. відносної (аналогичн.)  
пам'яті  
vs  
пам'яті відносної  
пам'яті



### GRU

### Уникуючий LSTM

Vanishing gradient: GRU



Vanishing gradient: GRU

$$\begin{aligned} u^{(t)} &= \sigma(W_u h^{(t-1)} + U_u x^{(t)} + b_u) \\ r^{(t)} &= \sigma(W_r h^{(t-1)} + U_r x^{(t)} + b_r) \\ \tilde{h}^{(t)} &= \tanh(W_t h^{(t-1)} + U_t x^{(t)} + b_t) \\ h^{(t)} &= (1 - u^{(t)}) h^{(t-1)} + u^{(t)} \tilde{h}^{(t)} \end{aligned}$$

Annotations explain the vanishing gradient problem:

- Forget gate: controls what is forgotten. If it is applied too often, previous hidden states are lost.
- Update gate: controls what new hidden states are used. If it is applied too often, current hidden states are lost.
- New hidden state: controls what new hidden states are generated. If it is applied too often, hidden states grow exponentially.
- Hidden state: controls what hidden states are output. If it is applied too often, hidden states grow exponentially.

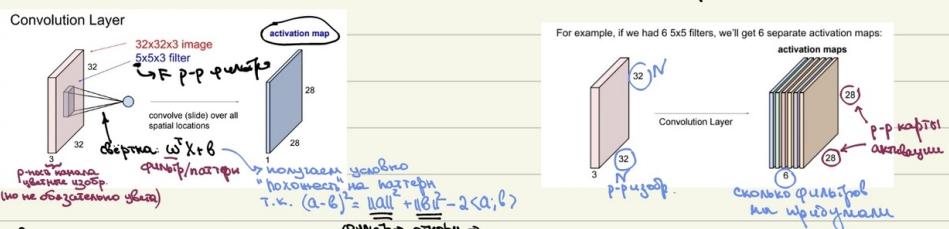
VIBR: GRU є заспокоюючим, не пакує відповідь/хоче.

=> Графічно с LSTM, но єдину хотіть дати, але GRU, який є логічною заміною.

### 30 Matrix convolution. Convolutional layer, backpropagation through it. Hyperparameters of Convs. 1x1 convolutions, comparison to FC layers. Max/Average Pooling.

30. Matrix convolution. Conv.layer, backprop.  
Hyperparameters of Convs, 1x1 conv., FC-layer  
Max/Average Pooling

Идея: хранение "сжатых" данных на картинке



Сверка: глб. окн.

Stride - шаг, S. Если  $S=1$ , то передвигаем все квадраты  $F \times F$

Pad - на сколько мы "отступаем" за границу.  
 Р-р карты акт.:  $\frac{N-F+2P}{S} + 1$  "рамка" из 0<sup>4</sup> при  $P=1$

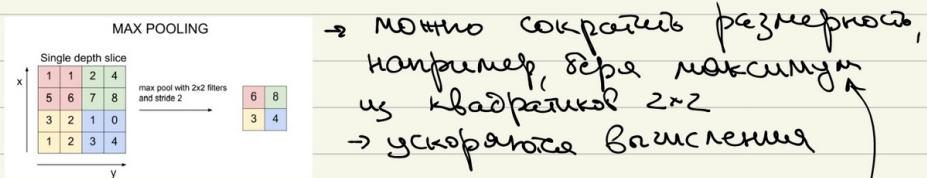
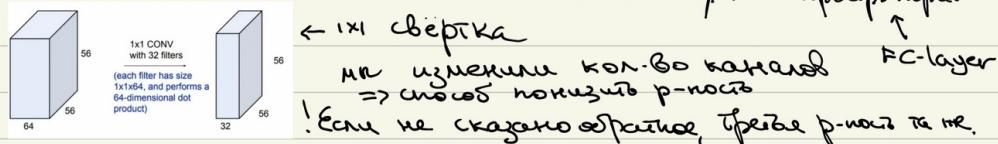
Ну и то, чтобы анализировать край + сохранение размерности

Пример:

|    |    |
|----|----|
| 32 | 32 |
| 32 | 32 |

10 фильтров  $5 \times 5$   $S=1 P=2$   $\Rightarrow$  Параллельно: за 8-смену  
 у каждого фильтра  $5 \cdot 5 \cdot 3 + 1 = 76$   
 + дополнительные на кон-бо:  $10 \cdot 76 = 760$

3 А если бы писали, то было бы  $\sim 32^4 \cdot 3 \cdot 10 > 760$   
 но мы сохранили простр. независимо



# 31 Main ideas of AlexNet, VGG, Inception (GoogLeNet), ResNet architectures.

**31 Main ideas of AlexNet, VGG, Inception (GoogLeNet), ResNet**

60-e roda. "zuma" nevronok + DO-IO-RE

**Le-Net 5**

[LeNet-5, LeCun 1998]

**AlexNet**

Alex Krizhevsky  
на соревнование Image Net  
2012 год

суть кипыя но хотят не Le-Net

**ImageNet experiments**

ZFNet, 2013

чтобы лучше защищать  
и чтобы лучше тренировать

=> чтобы лучше предсказывать,  
затем AlexNet (апп-па та же)

**VGG (2014)**

VGG Net имеет 2 раза лучше AlexNet  
много слоев и меньше FC  
+ max pooling

Однако много весов  
(93 МБ для классификации  
224x224)

**GoogLeNet 2014**

Inception module - способствует упрощению  
без потери важных размеров  
~6,7% ошибка

**ResNet 2015**

34-layer plain  
34-layer residual  
 $H(x) = F(x) + x$

id<sub>x</sub> => пределить не засыхает  
+ уменьшаете BatchNorm

## 32 Geometrical methods in ML. Clustering problem. IsoMap, LLE, DBSCAN, k-means, t-SNE