

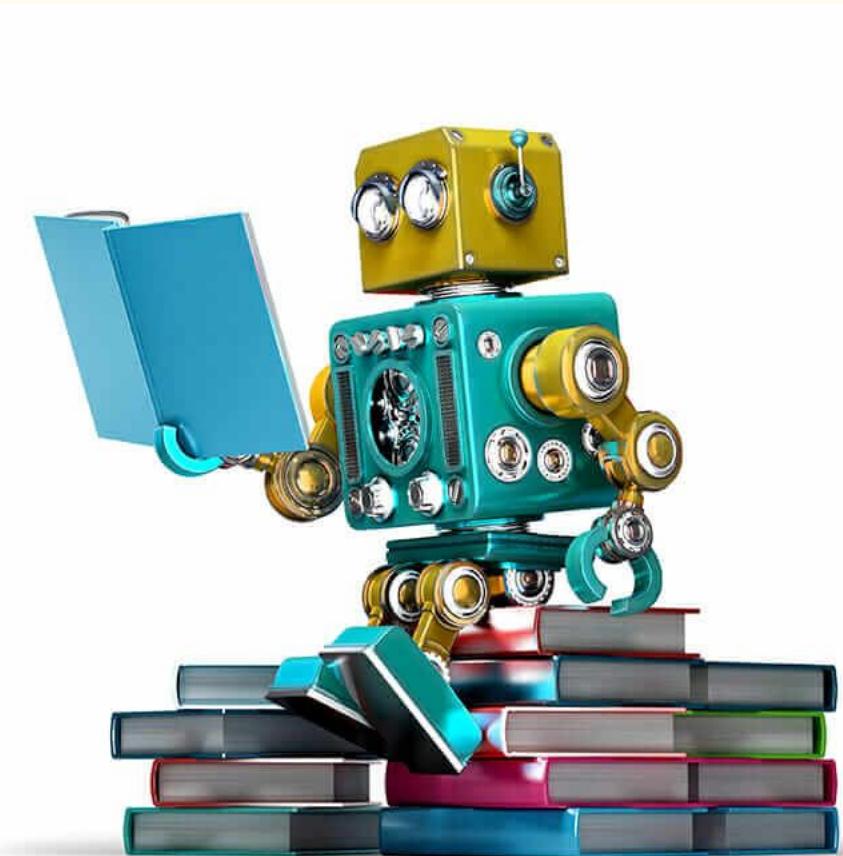
Модели, ансамбли моделей

Татьяна Гайнцева
МФТИ, ШАД, DL School

@atmyre

План

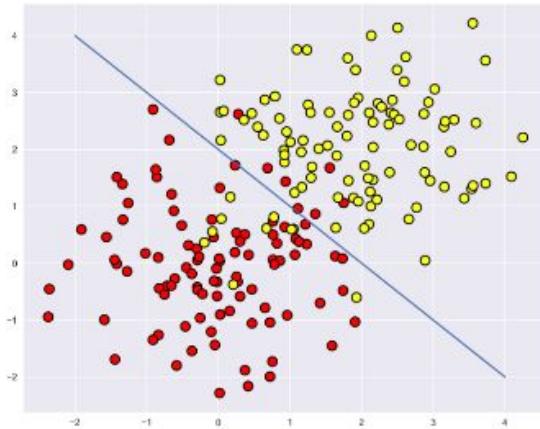
1. Зоопарк моделей
 - a. Обзор
 - b. Линейные модели
 - i. Линейная регрессия
 - ii. Логистическая регрессия
 - iii. SVM
 - c. Логические алгоритмы классификации
 - i. Закономерность и информативность
 - ii. Решающие деревья
2. Композиции алгоритмов
 - a. Градиентный бустинг (AnyBoost, XGBoost)
 - b. Простое голосование
 - c. Смеси алгоритмов
3. Scikit-Learn



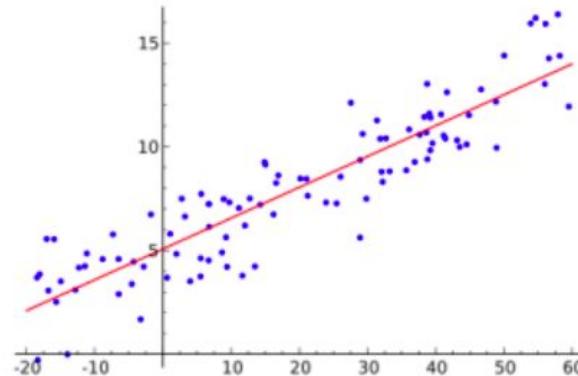
Модели



Регрессия VS Классификация

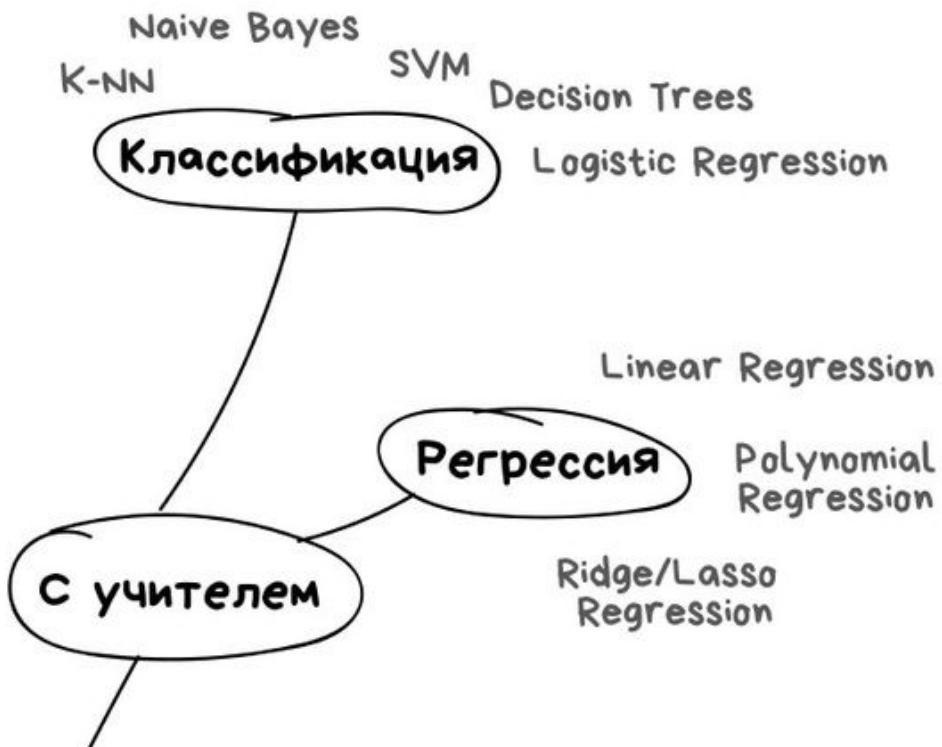


$$Y = \{1, \dots, N\}$$



$$Y \subseteq \mathbf{R}$$





Наивный байесовский классификатор

- Spam detection
- Сегментация новостных статей по их тематике;
- Определение эмоционального окраса блока текста;
- Программное обеспечение для распознавания лиц.

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

метка класса

$$c = \arg \max_C P(C|O)$$

объект для классификации

$$c = \arg \max_{c \in C} P(c|o_1 o_2 \dots o_n) = \arg \max_{c \in C} P(c) \prod P(o_i|c)$$

```
from __future__ import division
from collections import defaultdict
from math import log

def train(samples):
    classes, freq = defaultdict(lambda:0), defaultdict(lambda:0)
    for feats, label in samples:
        classes[label] += 1                      # count classes frequencies
        for feat in feats:
            freq[label, feat] += 1                # count features frequencies

    for label, feat in freq:                     # normalize features frequencies
        freq[label, feat] /= classes[label]
    for c in classes:                          # normalize classes frequencies
        classes[c] /= len(samples)

    return classes, freq                       # return P(C) and P(O|C)
```

подсчет
параметров

```
def classify(classifier, feats):
    classes, prob = classifier
    return min(classes.keys(),           # calculate argmin(-log(C|O))
               key = lambda cl: -log(classes[cl]) + \
                           sum(-log(prob.get((cl,feat), 10**(-7))) for feat in feats))
```

расчет
формулы

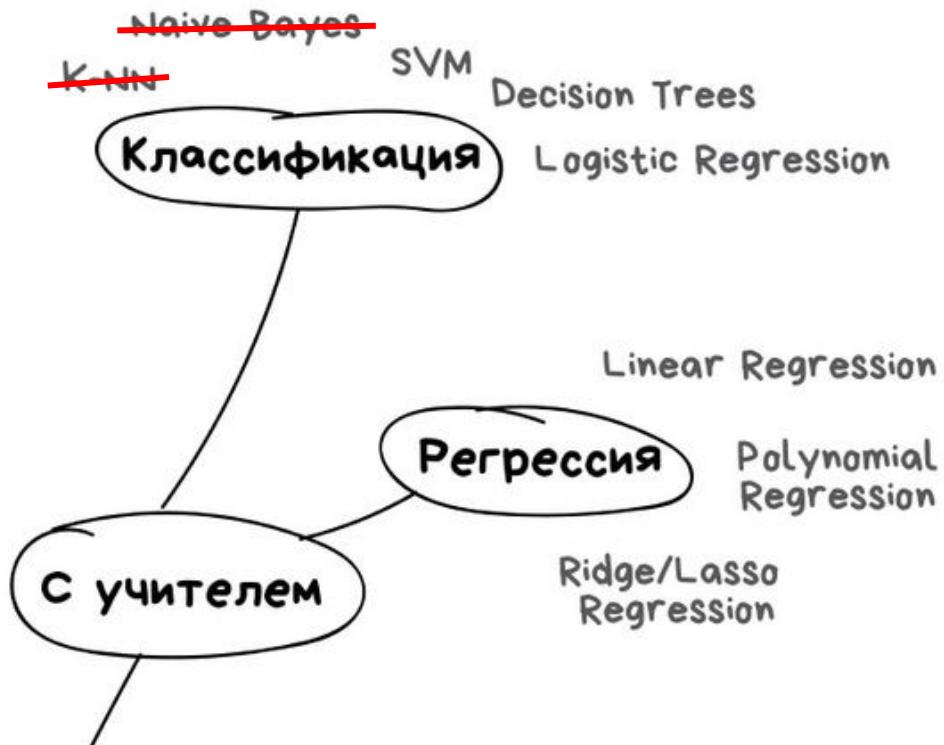
Найвный Байес в SKLearn

GaussianNB: implements the Gaussian Naive Bayes algorithm for classification. The likelihood of the features is assumed to be Gaussian

MultinomialNB: implements the naive Bayes algorithm for multinomially distributed data

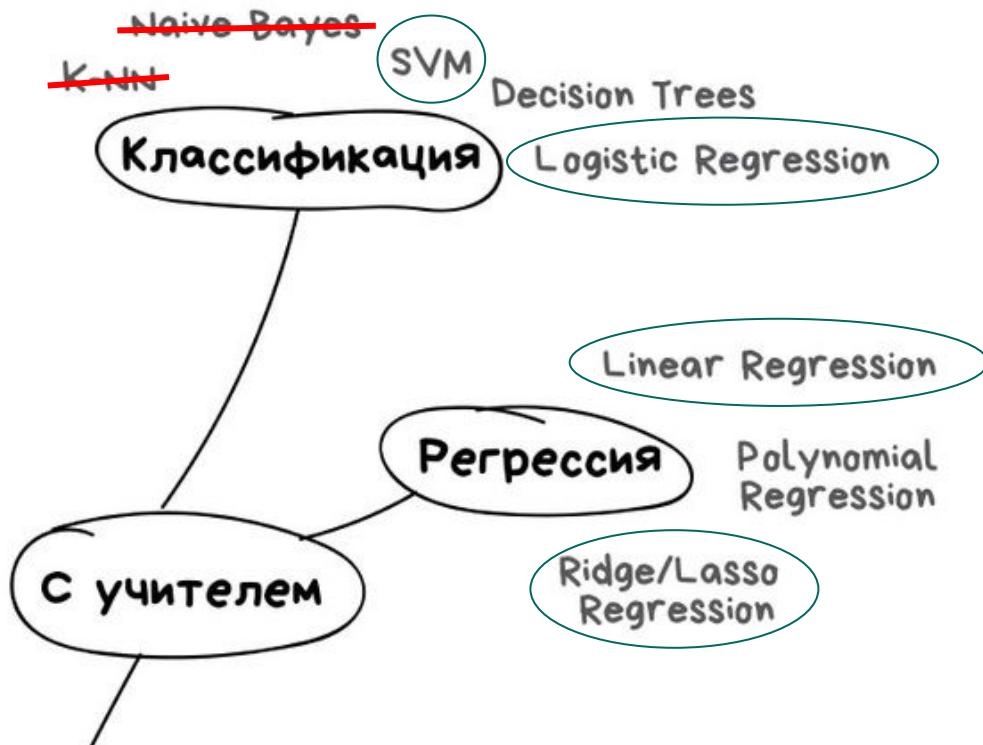
ComplementNB: implements the complement naive Bayes (CNB) algorithm. CNB is an adaptation of the standard multinomial naive Bayes (MNB) algorithm that is particularly suited for imbalanced data sets.

BernoulliNB: implements the naive Bayes training and classification algorithms for data that is distributed according to multivariate Bernoulli distributions

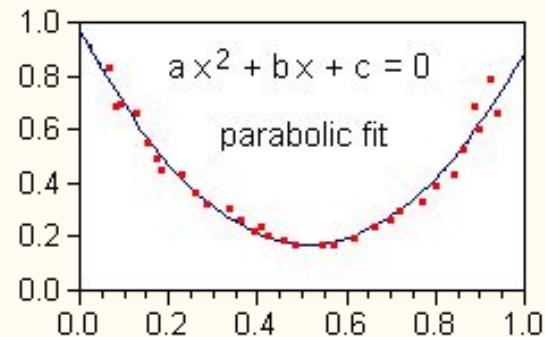
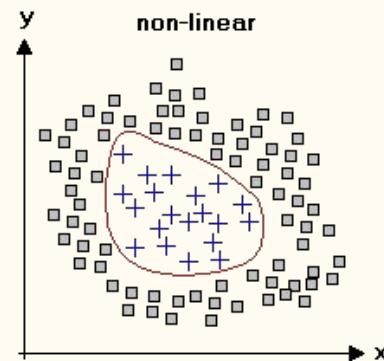
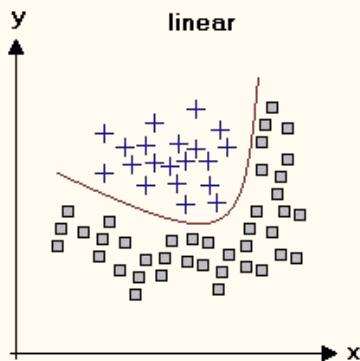
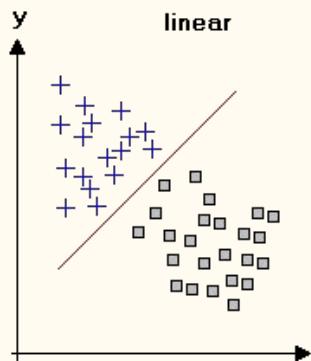


Линейные модели



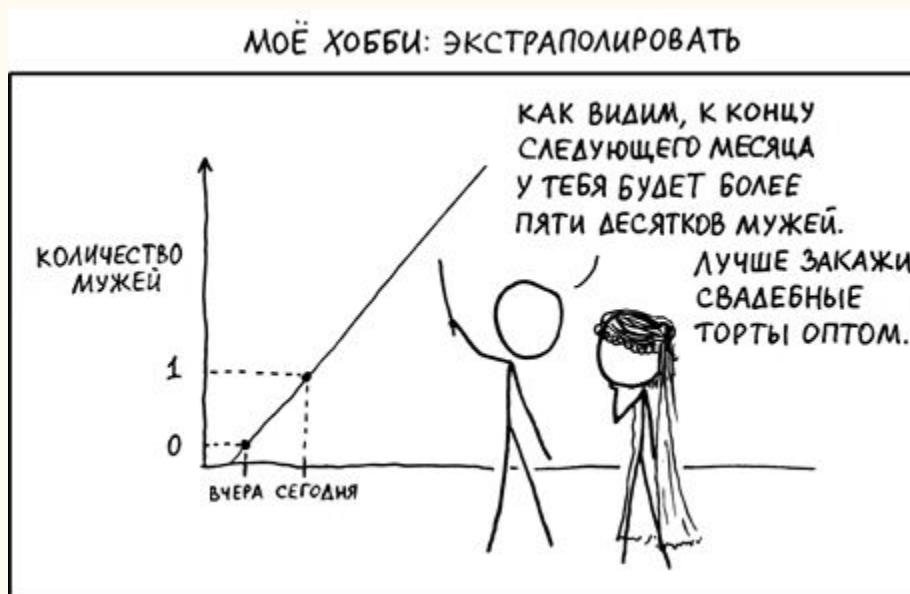


Линейная модель



$$Y_i = \beta_0 + \beta_1 \phi_1(X_{i1}) + \cdots + \beta_p \phi_p(X_{ip}) + \varepsilon_i \quad i = 1, \dots, n$$

Линейная регрессия

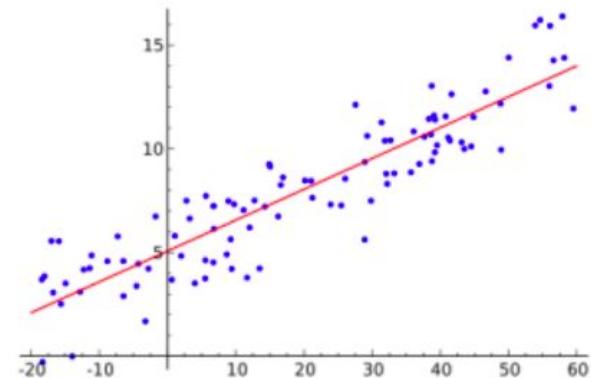


Линейная регрессия

$$\vec{y} = X\vec{w} + \epsilon,$$

где

- $\vec{y} \in \mathbb{R}^n$ – объясняемая (или целевая) переменная;
- w – вектор параметров модели (в машинном обучении эти параметры часто называют весами);
- X – матрица наблюдений и признаков размерности n строк на $m + 1$ столбцов (включая фиктивную единичную колонку слева) с полным рангом по столбцам: $\text{rank}(X) = m + 1$;
- ϵ – случайная переменная, соответствующая случайной, непрогнозируемой ошибке модели.



Линейная регрессия

Постановка задачи:

$$y = WX$$

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} & 1 \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{d1} & x_{d2} & x_{d3} & \dots & x_{dn} & 1 \end{bmatrix}$$

Решение:

$$W_{ans} = (W^T W)^{-1} W^T y$$

Anyone can see a problem?

Линейная регрессия

Решение проблемы: Регуляризация

$$W_{ans} = (W^T W + \lambda I)^{-1} W^T y$$

I -- единичная матрица

Линейная регрессия

Градиентный спуск:

$$y = WX$$

Постановка задачи:

$$X = \begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} & 1 \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} & 1 \\ \vdots & \vdots & \vdots & \ddots & \vdots & \vdots \\ x_{d1} & x_{d2} & x_{d3} & \dots & x_{dn} & 1 \end{bmatrix}$$

1. Инициализация W случайными значениями
2. Итеративно для всех сэмплов x из X_{train} :
 3. Вычислить функцию потерь на x
 4. Вычислить производные $grad_w$ по каждому w из W
 5. Обновить веса $w = w - lr * grad_w$
 6. Полученное W = (w) искомое

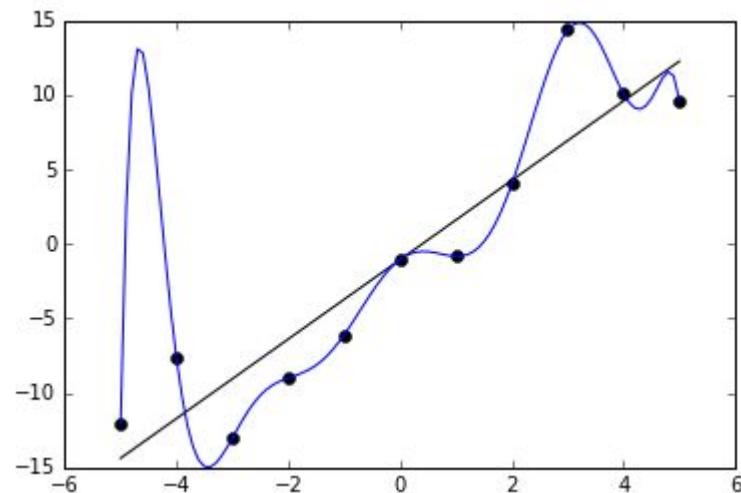
Функция потерь для линейной регрессии:

$$MSE = \frac{1}{N} \sum (y_i - (Wx_i + b))^2$$

Линейная регрессия

При решении задачи регрессии методом *градиентного спуска*:

Переобучение:



Как бороться?

Линейная регрессия

Регуляризация!

L1 (Lasso regression):

$$MSE = \frac{1}{N} \sum (y_i - (Wx_i + b))^2 + \lambda \sum |W_i|$$

L2 (Ridge regression):

$$MSE = \frac{1}{N} \sum (y_i - (Wx_i + b))^2 + \lambda \sum (W_i)^2$$

SVM

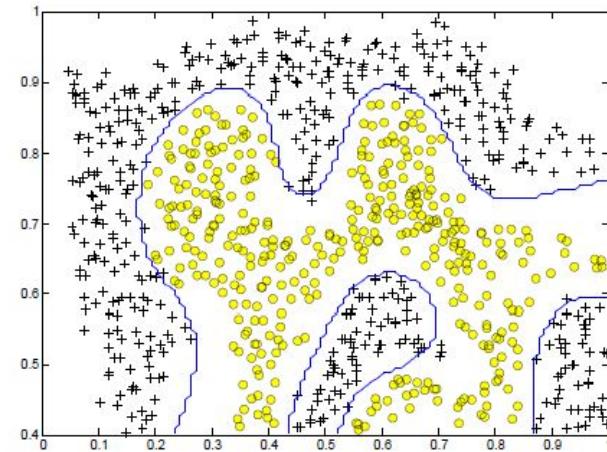
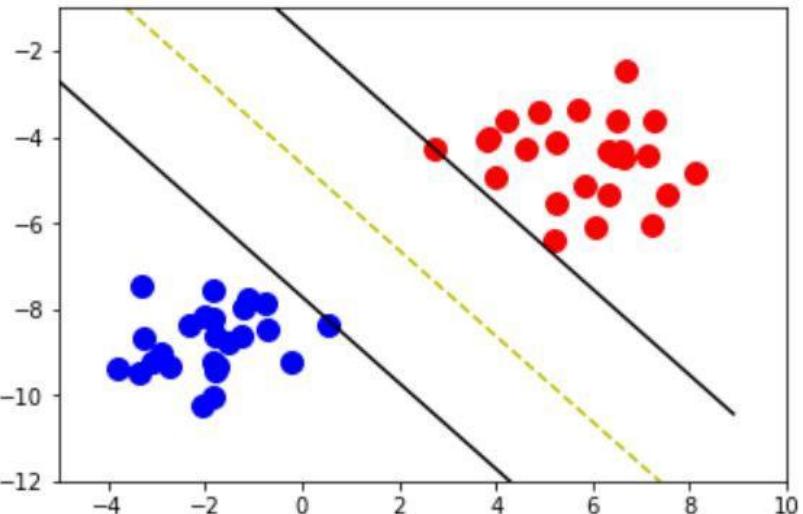


Figure 5: SVM (Gaussian Kernel) Decision Boundary (Example Dataset 2)

<http://www.machinelearning.ru/wiki/index.php?title=SVM>

Логистическая регрессия

Линейная регрессия:

$$y = WX$$

Логистическая регрессия:

$$y = \sigma(WX), \quad \sigma = \frac{1}{1 + e^{-x}}$$

Решающее правило (модель):

$$a(x, w) = \text{sign} \left(\sum_{j=1}^n w_j f_j(x) - w_0 \right) = \text{sign} \langle x, w \rangle,$$

http://www.machinelearning.ru/wiki/index.php?title=%D0%A0%D0%BE%D0%BD%D0%B8%D0%BC%D0%B8%D1%82%D0%BD%D0%BE%D0%B3%D0%BE_%D0%BC%D0%BE%D0%BF%D0%BB%D0%BE%D0%BD%D0%BD%D0%BE%D0%BC

Логические методы



Логические методы классификации

Логическая закономерность – предикат $R:X \rightarrow \{0,1\}$:

1. Интерпретируем
 - a. записывается на естественном языке
 - b. зависит от небольшого числа параметров (1-7)
2. Информативен

$$p_c(R) = \#\{x_i : R(x_i)=1 \text{ и } y_i=c\} \rightarrow \max;$$

$$n_c(R) = \#\{x_i : R(x_i)=1 \text{ и } y_i \neq c\} \rightarrow \min;$$



Логические методы классификации

Примеры информативности:

*Если возраст > 60 и пациент ранее перенёс инфаркт,
то операцию не делать, риск отрицательного исхода 60%.*

*Если в анкете указан домашний телефон
и зарплата > \$2000 и сумма кредита < \$5000
то кредит можно выдать, риск дефолта 5%.*

Закономерности: наборы правил

1. Пороговое условие (решающий пень, decision stump):

$$R(x) = [f_j(x) \leq a_j] \text{ или } [a_j \leq f_j(x) \leq b_j].$$

2. Конъюнкция пороговых условий:

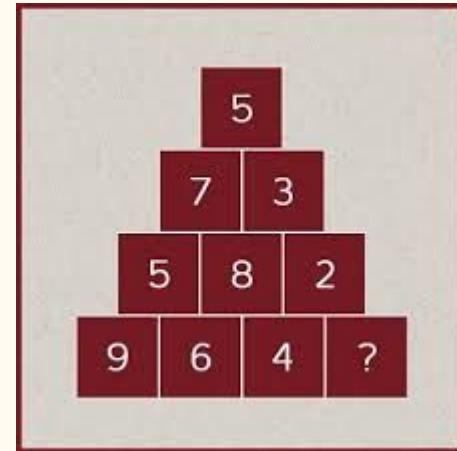
$$R(x) = \bigwedge_{j \in J} [a_j \leq f_j(x) \leq b_j].$$

3. Синдром — выполнение не менее d условий из $|J|$,
(при $d = |J|$ это конъюнкция, при $d = 1$ — дизъюнкция):

$$R(x) = \left[\sum_{j \in J} [a_j \leq f_j(x) \leq b_j] \geq d \right],$$

Обучение логических классификаторов

1. Шаги индукции правил (rule induction)
 - a. Выбор семейства правил для поиска закономерностей
 - b. Порождение правил (rule generation)
 - c. Отбор правил-закономерностей (rule selection)
 - d. Построение классификатора из правил как из признаков. Например, взвешенное голосование.



- ❶ Как изобретать признаки?
 - не наука, а искусство (озарения, мозговые штурмы,...)
- ❷ Какого вида закономерности нам нужны?
 - простые формулы от малого числа признаков
- ❸ Как определять информативность?
 - так, чтобы одновременно $p_c \rightarrow \max$, $n_c \rightarrow \min$
- ❹ Как строить отдельные закономерности?
 - методами отбора признаков
- ❺ Как объединять закономерности в алгоритм?
 - методами построения композиций классификаторов

Закономерность — это хорошо интерпретируемый одноклассовый классификатор с отказами.

Оценка качества информативности

Проблема: надо сравнивать закономерности R .

Как свернуть два критерия в один критерий информативности?

$$\begin{cases} p(R) \rightarrow \max \\ n(R) \rightarrow \min \end{cases} \stackrel{?}{\Rightarrow} I(p, n) \rightarrow \max$$

Очевидные, но не всегда адекватные свёртки:

- $I(p, n) = \frac{p}{p + n} \rightarrow \max$ (precision);
- $I(p, n) = p - n \rightarrow \max$ (accuracy);

Пример:

при $P = 200$, $N = 100$ и различных p и n .

Простые эвристики не всегда адекватны:

p	n	$p-n$	$p-5n$	$\frac{p}{P}-\frac{n}{N}$	$\frac{p}{n+1}$	IStat· ℓ	IGain· ℓ	$\sqrt{p}-\sqrt{n}$
50	0	50	50	0.25	50	22.65	23.70	7.07
100	50	50	-150	0	1.96	2.33	1.98	2.93
50	9	41	5	0.16	5	7.87	7.94	4.07
5	0	5	5	0.03	5	2.04	3.04	2.24
100	0	100	100	0.5	100	52.18	53.32	10.0
140	20	120	40	0.5	6.67	37.09	37.03	7.36

Более адекватные свертки

сколько информации мы получим о разделении объектов на два класса, если узнаем \mathbf{R} ?

- энтропийный критерий прироста информации:

$$\text{IGain}(p, n) = h\left(\frac{P}{\ell}\right) - \frac{p+n}{\ell} h\left(\frac{p}{p+n}\right) - \frac{\ell-p-n}{\ell} h\left(\frac{P-p}{\ell-p-n}\right) \rightarrow \max,$$

где $h(q) = -q \log_2 q - (1-q) \log_2(1-q)$

- критерий Джини (Gini impurity):

$$\text{IGini}(p, n) = \text{IGain}(p, n) \text{ при } h(q) = 4q(1-q)$$

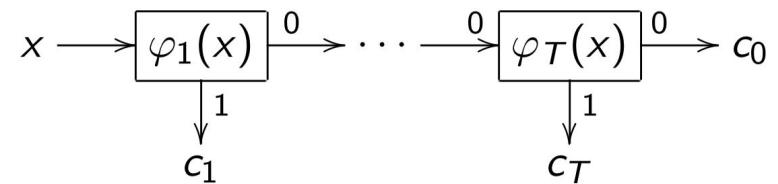
“загрязненность” распределения

Композиции закономерностей

- Взвешенное (или простое) голосование

$$a(x) = \arg \max_{y \in Y} \sum_{t=1}^{T_y} w_{yt} R_{yt}(x)$$

- Решающий список (по старшинству)



Резюме до этого момента

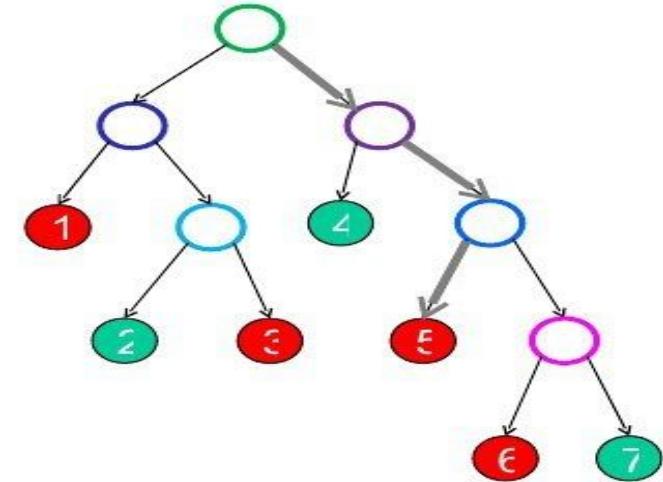
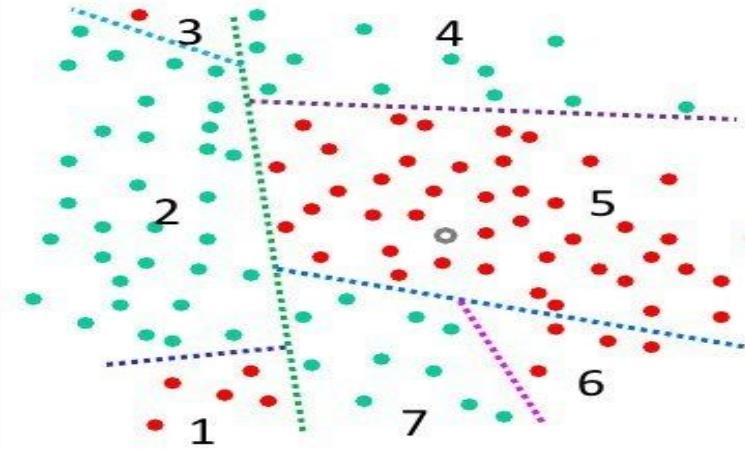
- Правило -- интерпретируемый предикат (функция) $X: - \rightarrow \{0, 1\}$
- Закономерность -- информативное правило (набор правил)
- Критерий информативности существует много разных
- Как строить закономерность: отбирать признаки по их информативности
- Как строить композиции закономерностей:
 - голосование
 - решающий список
 - *решающее дерево*

Решающее дерево



Пример решающего дерева

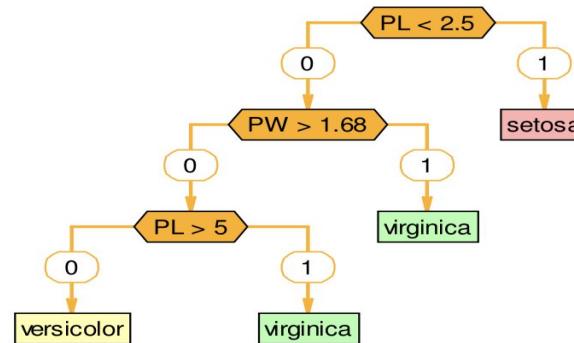
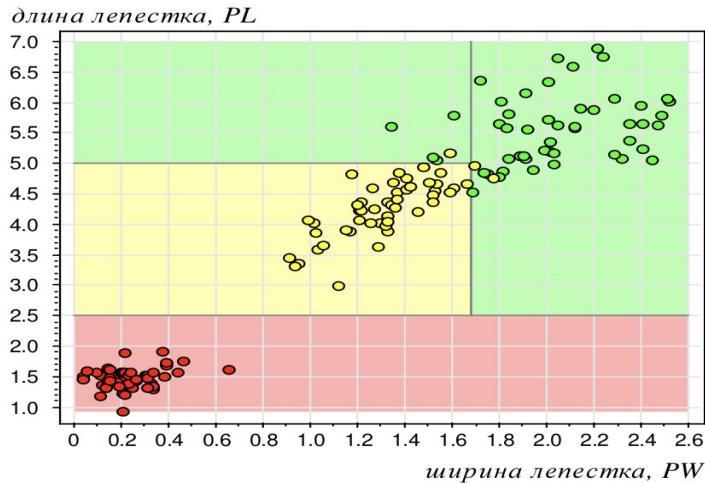
Решающее дерево



Slide by V.Lempitsky

Пример решающего дерева

Задача Фишера о классификации цветков ириса на 3 класса, в выборке по 50 объектов каждого класса, 4 признака.



На графике: в осях двух самых информативных признаков (из 4) два класса разделились без ошибок, на третьем 3 ошибки.

Построение решающего дерева

- 1: **ПРОЦЕДУРА** LearnID3 ($U \subseteq X^\ell$);
- 2: **если** все объекты из U лежат в одном классе $c \in Y$ **то**
- 3: **вернуть** новый лист v , $c_v := c$;
- 4: **найти** предикат с максимальной информативностью:
 $\beta := \arg \max_{\beta \in \mathcal{B}} I(\beta, U)$;
- 5: **разбить** выборку на две части $U = U_0 \cup U_1$ по предикату β :
 $U_0 := \{x \in U : \beta(x) = 0\}$;
 $U_1 := \{x \in U : \beta(x) = 1\}$;
- 6: **если** $U_0 = \emptyset$ **или** $U_1 = \emptyset$ **то**
- 7: **вернуть** новый лист v , $c_v := \text{Мажоритарный класс}(U)$;
- 8: **создать** новую внутреннюю вершину v : $\beta_v := \beta$;
построить левое поддерево: $L_v := \text{LearnID3}(U_0)$;
построить правое поддерево: $R_v := \text{LearnID3}(U_1)$;
- 9: **вернуть** v ;

Решающие деревья: + и -

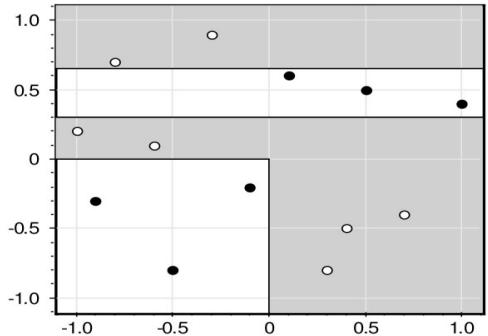
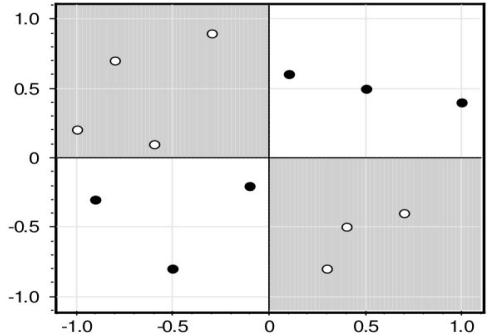
+

- Интерпретируемость, простота
- Гибкость: можно менять множества закономерностей
- Допустимы данные с пропусками
- Сложность построения линейна по длины выборки

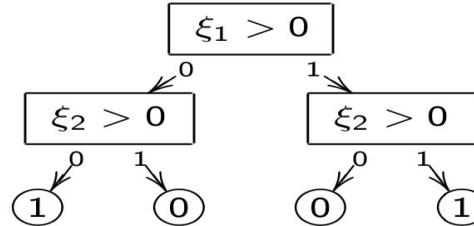
-

- Жадный алгоритм выбора правила для вершины пере усложняет дерево
- Чем дальше вершина от корня, тем слабее статистическая надежность решающего правила
- Высокая чувствительность к шуму, составу выборки

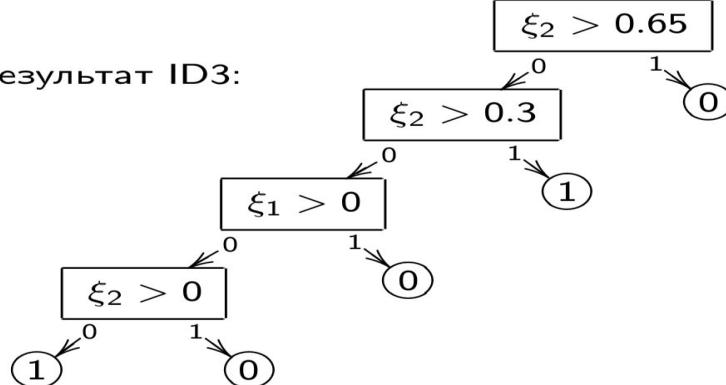
Проблема переобучения



Оптимальное дерево для задачи XOR:



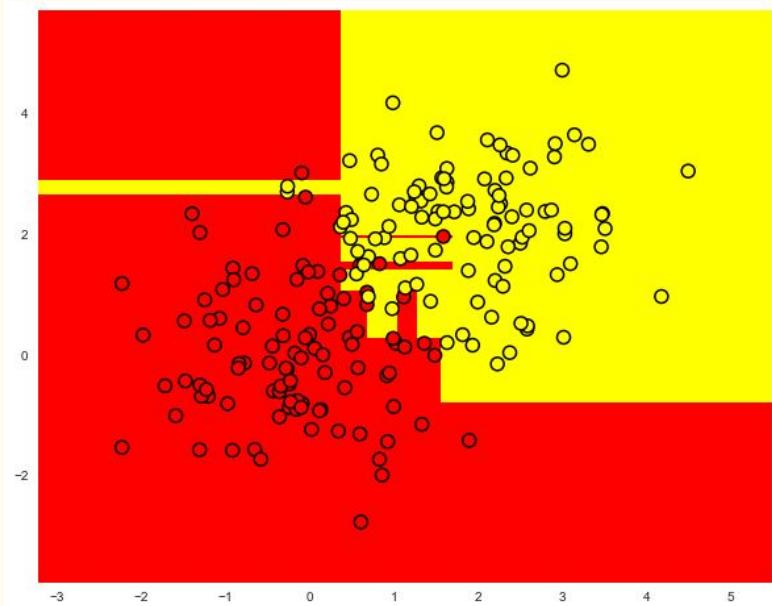
Результат ID3:



Проблема переобучения

Пути решения:

1. Ограничение глубины дерева
2. Ограничение минимального числа объектов в листе
3. Стрижка дерева (pruning)



sklearn.tree.DecisionTreeClassifier

```
class sklearn.tree. DecisionTreeClassifier (criterion='gini', splitter='best', max_depth=None,  
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features=None, random_state=None,  
max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, class_weight=None, presort=False)
```

[source]

A decision tree classifier.

Read more in the [User Guide](#).

Parameters: **criterion** : string, optional (default="gini")

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

splitter : string, optional (default="best")

The strategy used to choose the split at each node. Supported strategies are "best" to choose the best split and "random" to choose the best random split.

max_depth : int or None, optional (default=None)

The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

Обобщение на случай регрессии: $Y = \mathbb{R}$, $y_v \in \mathbb{R}$.

U — множество объектов x_i , дошедших до вершины v

Мера неопределённости — среднеквадратичная ошибка

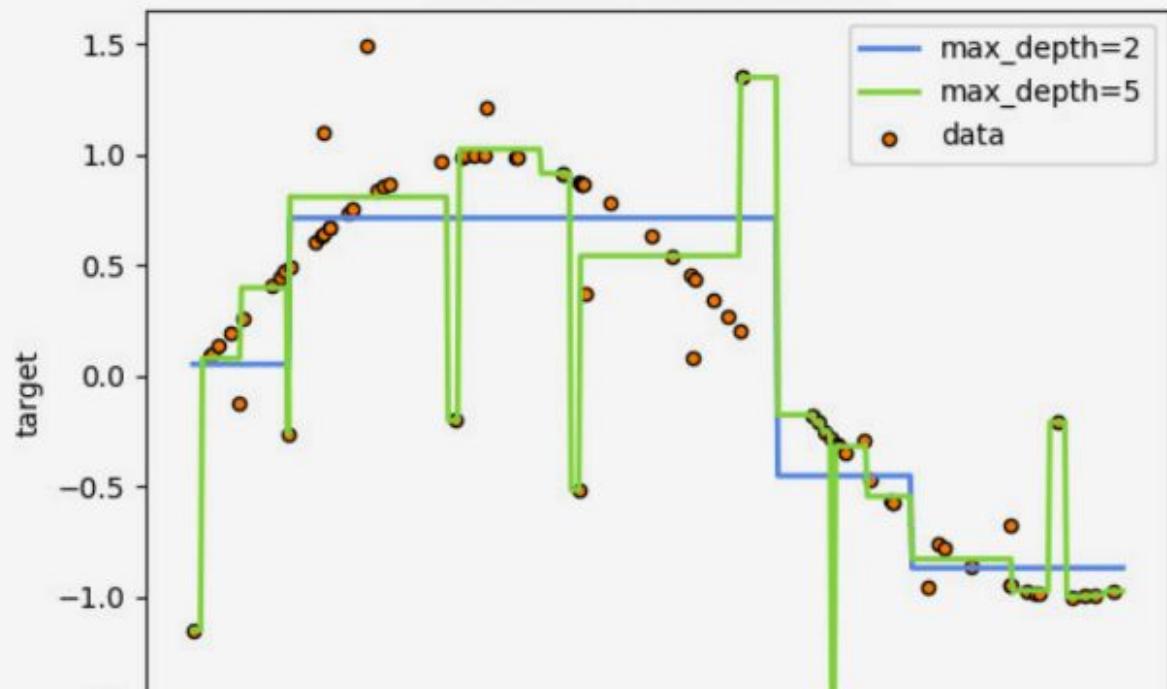
$$\Phi(U) = \min_{y \in Y} \frac{1}{|U|} \sum_{x_i \in U} (y - y_i)^2$$

Значение y_v в терминальной вершине v — МНК-решение:

$$y_v = \frac{1}{|U|} \sum_{x_i \in U} y_i$$

Дерево регрессии $a(x)$ — это кусочно-постоянная функция.

Decision Tree Regression

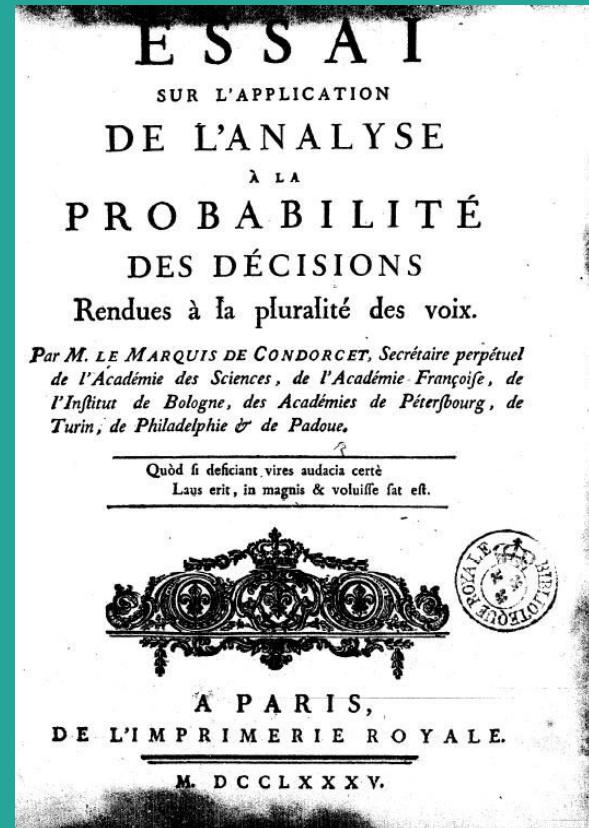


Композиции алгоритмов

100
010

Если каждый член жюри имеет независимое мнение, и если вероятность правильного решения члена жюри больше 0.5 , то тогда вероятность правильного решения присяжных в целом возрастает с увеличением количества членов жюри и стремится к единице.

Если же вероятность быть правым у каждого из членов жюри меньше 0.5 , то вероятность принятия правильного решения присяжными в целом монотонно уменьшается и стремится к нулю с увеличением количества присяжных.



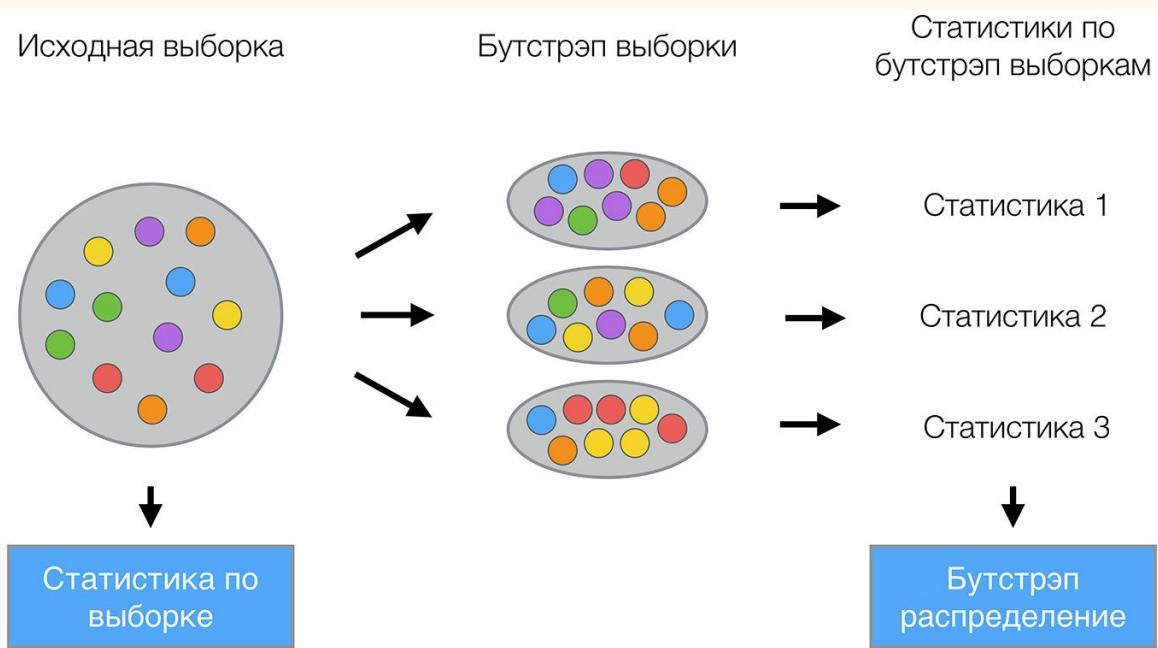
принцип Кондорсе, 1784

Собралось около 800 человек, которые попытались угадать вес быка на ярмарке. Бык весил 1198 фунтов. Ни один крестьянин не угадал точный вес быка, но если посчитать среднее от их предсказаний, то получим 1197 фунтов.



Гальтон, 1906 год

Bootstrap



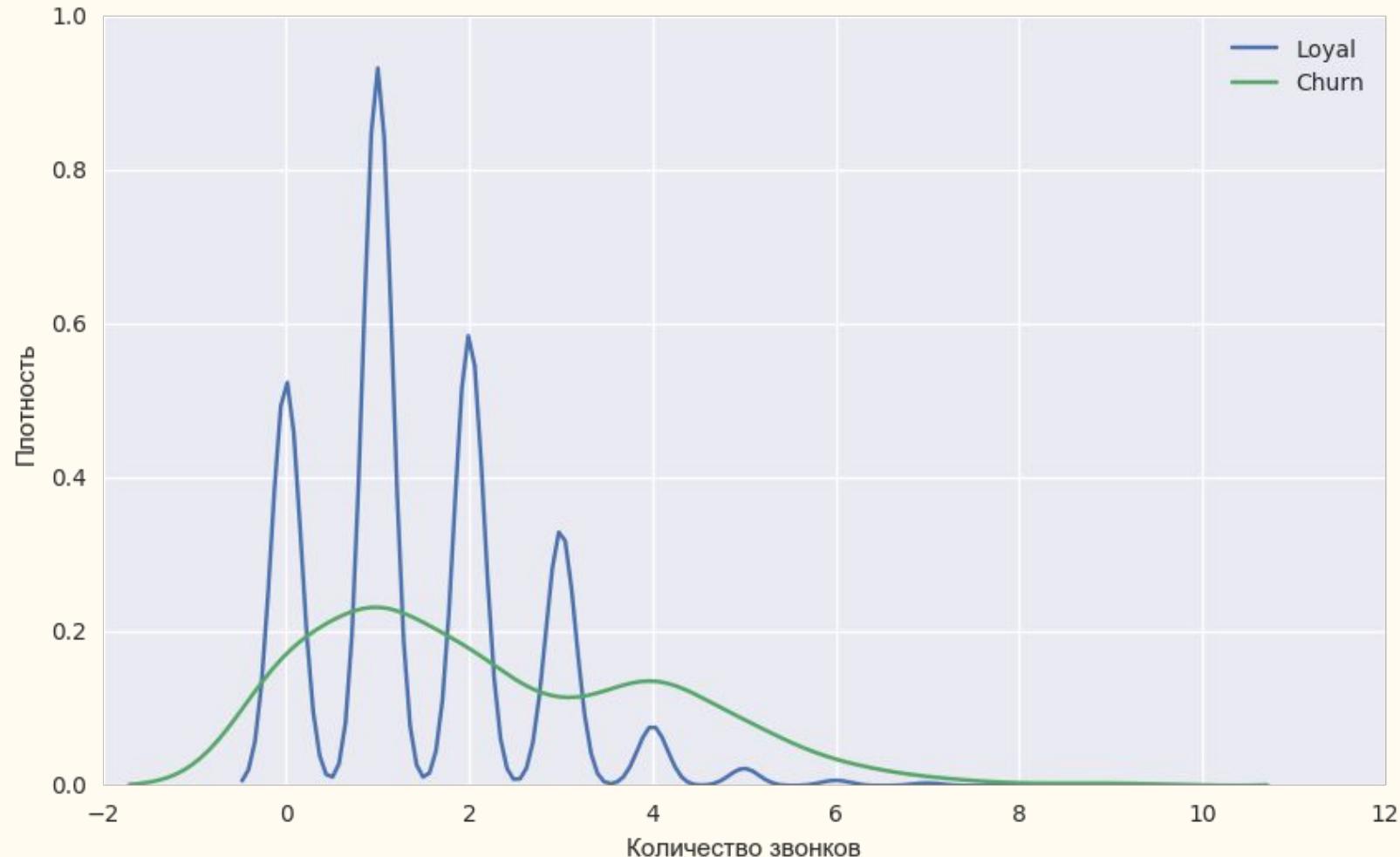
Бинарная классификация оттока клиентов

```
import pandas as pd
from matplotlib import pyplot as plt
plt.style.use('ggplot')
plt.rcParams['figure.figsize'] = 10, 6
import seaborn as sns
%matplotlib inline

telecom_data = pd.read_csv('data/telecom_churn.csv')

fig = sns.kdeplot(telecom_data[telecom_data['Churn'] == False]['Customer service calls'], label = 'Loyal')
fig = sns.kdeplot(telecom_data[telecom_data['Churn'] == True]['Customer service calls'], label = 'Churn')
fig.set(xlabel='Количество звонков', ylabel='Плотность')
plt.show()
```

Мало данных, одна из главных фич — количество звонков в сервисный центр



Оценим, сколько в среднем делает звонков каждая из групп.

Данных мало, поэтому искать среднее не совсем правильно, применим **bootstap** и сделаем интервальную оценку среднего

```
import numpy as np
def get_bootstrap_samples(data, n_samples):
    # функция для генерации подвыборок с помощью бутстрэпа
    indices = np.random.randint(0, len(data), (n_samples, len(data)))
    samples = data[indices]
    return samples
def stat_intervals(stat, alpha):
    # функция для интервальной оценки
    boundaries = np.percentile(stat, [100 * alpha / 2., 100 * (1 - alpha / 2.)])
    return boundaries
```

Оценим, сколько в среднем делает звонков каждая из групп.

Данных мало, поэтому искать среднее не совсем правильно, применим **bootstrap** и сделаем интервальную оценку среднего

```
import numpy as np
def get_bootstrap_samples(data, n_samples):
    # функция для генерации подвыборок с помощью бутстрэпа
    indices = np.random.randint(0, len(data), (n_samples, len(data)))
    samples = data[indices]
    return samples
def stat_intervals(stat, alpha):
    # функция для интервальной оценки
    boundaries = np.percentile(stat, [100 * alpha / 2., 100 * (1 - alpha / 2.)])
    return boundaries
```

```
# сохранение в отдельные патру массивы данных по лояльным и уже бывшим клиентам
loyal_calls = telecom_data[telecom_data['Churn'] == False]['Customer service calls'].values
churn_calls = telecom_data[telecom_data['Churn'] == True]['Customer service calls'].values

# ставим seed для воспроизводимости результатов
np.random.seed(0)

# генерируем выборки с помощью бутстрэпа и сразу считаем по каждой из них среднее
loyal_mean_scores = [np.mean(sample)
                      for sample in get_bootstrap_samples(loyal_calls, 1000)]
churn_mean_scores = [np.mean(sample)
                      for sample in get_bootstrap_samples(churn_calls, 1000)]
```

```
# генерируем выборки с помощью бутстрэра и сразу считаем по каждой из них среднее
loyal_mean_scores = [np.mean(sample)
                      for sample in get_bootstrap_samples(loyal_calls, 1000)]
churn_mean_scores = [np.mean(sample)
                      for sample in get_bootstrap_samples(churn_calls, 1000)]

# выводим интервальную оценку среднего
print("Service calls from loyal: mean interval", stat_intervals(loyal_mean_scores,
0.05))
print("Service calls from churn: mean interval", stat_intervals(churn_mean_scores,
0.05))
```

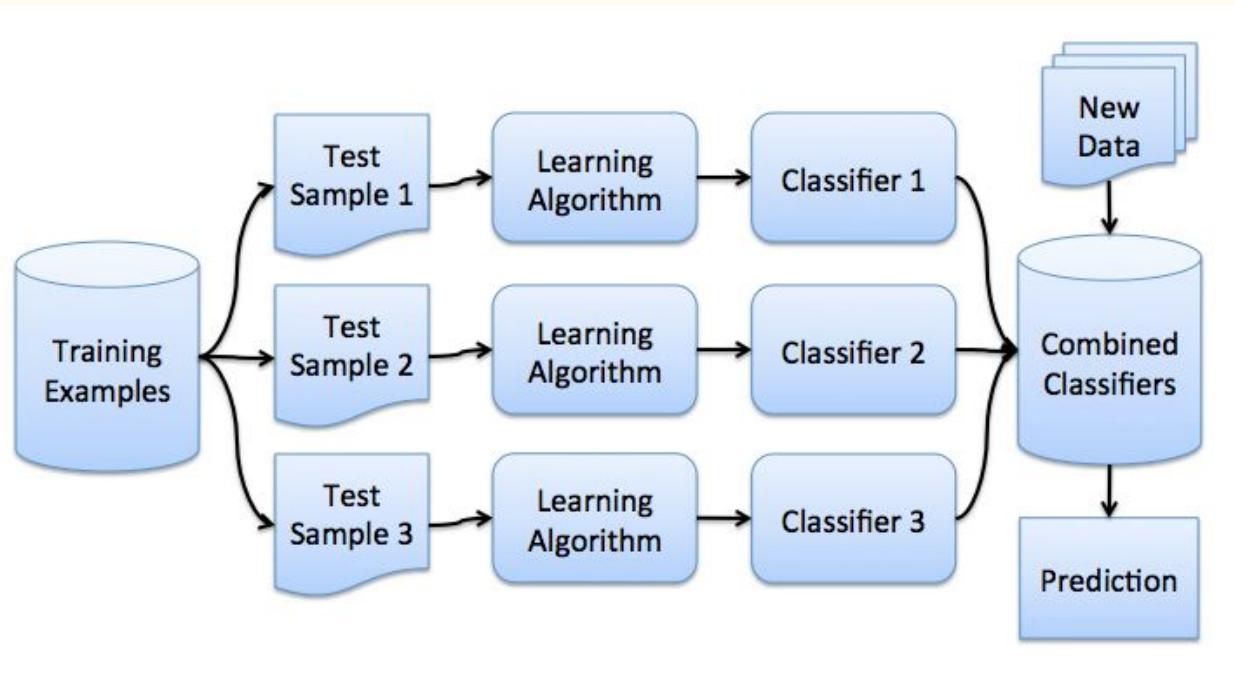
С 95% вероятностью среднее число звонков от лояльных клиентов будет лежать в промежутке между 1.40 и 1.50, в то время как наши бывшие клиенты звонили в среднем от 2.06 до 2.40 раз

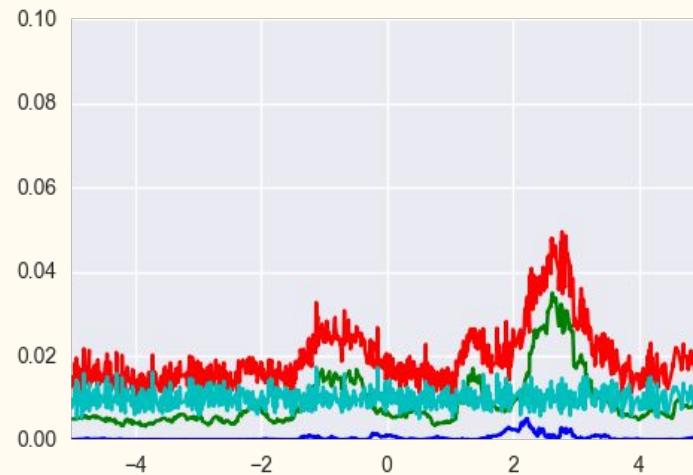
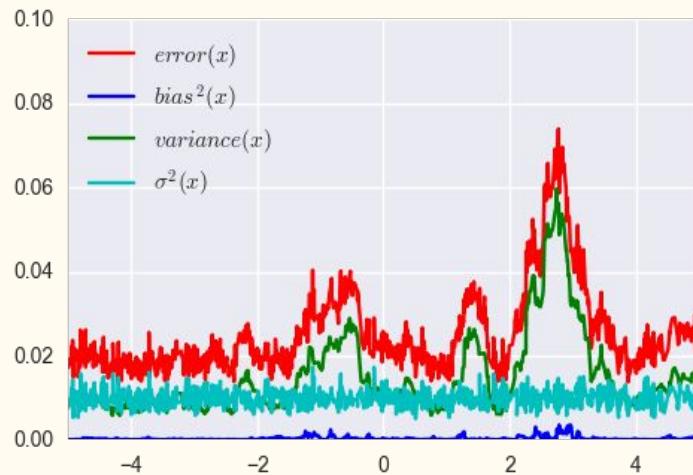
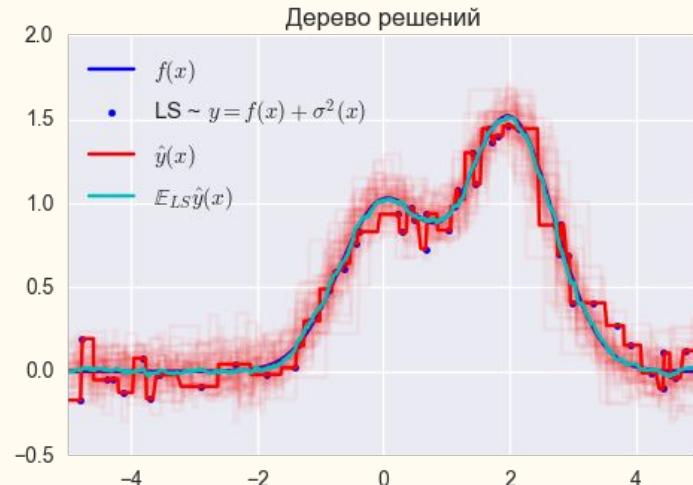
Bagging (bootstrap aggregating)

Дана train sample X. С помощью **bootstrap** сгенерируем из неё M выборок. Теперь на каждой выборке обучим свой классификатор.

Итоговый классификатор будет усреднять ответы всех этих алгоритмов:

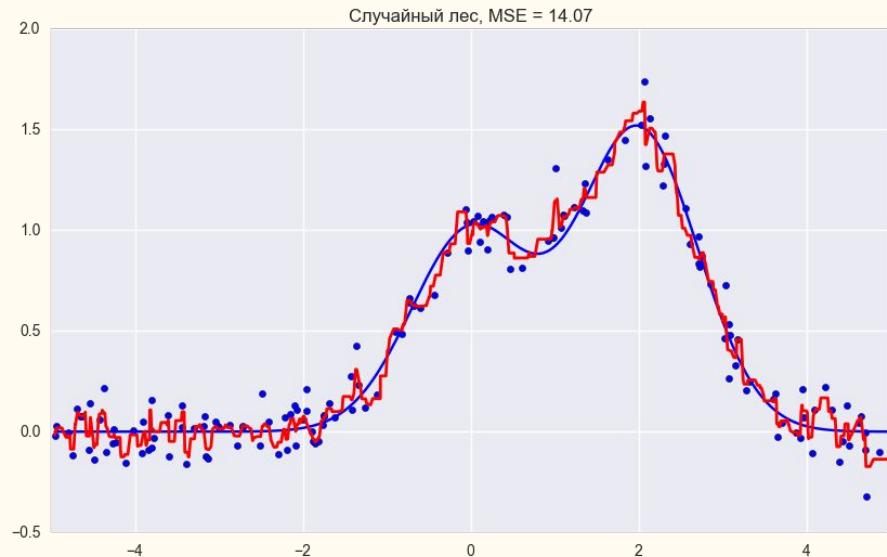
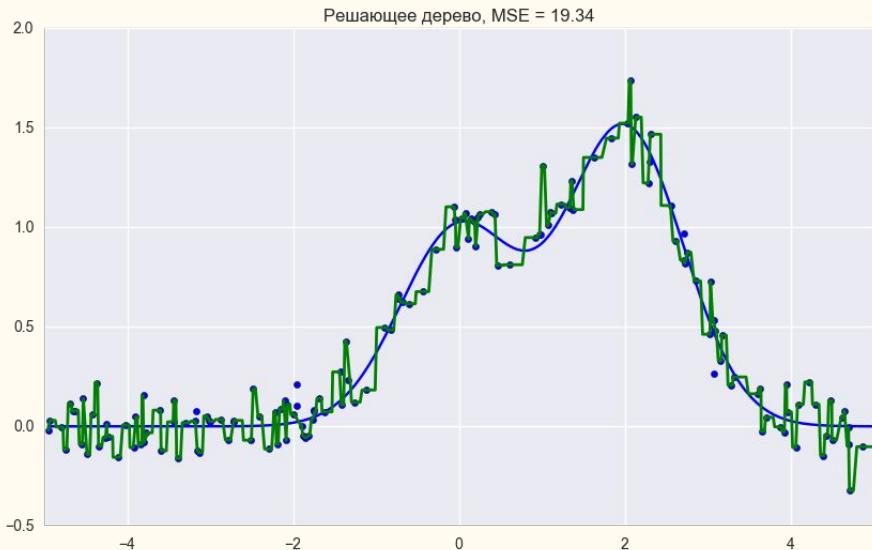
$$a(x) = \frac{1}{M} \sum_{i=1}^M a_i(x)$$



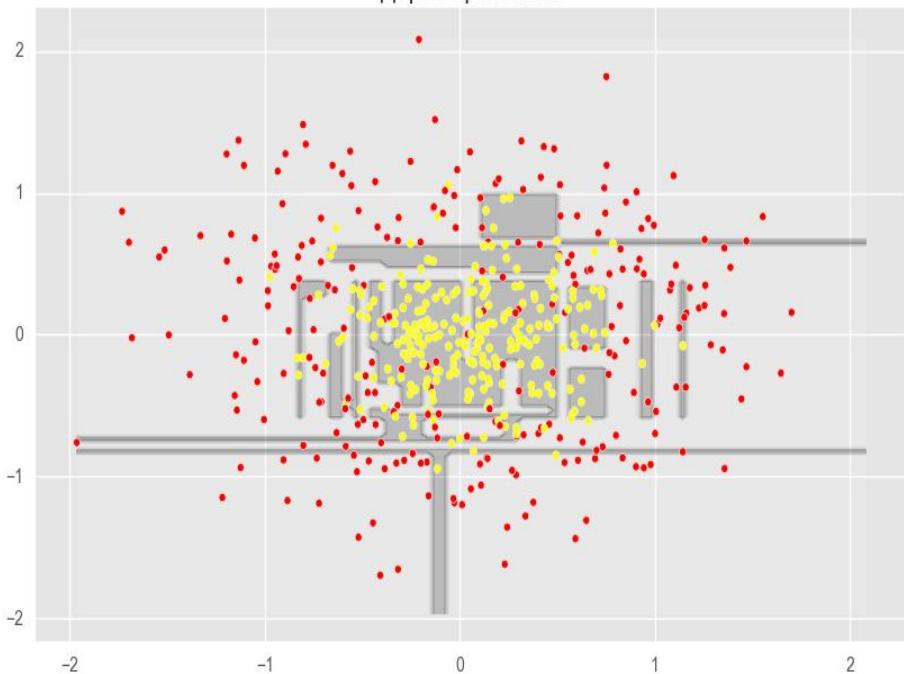


Random Forest

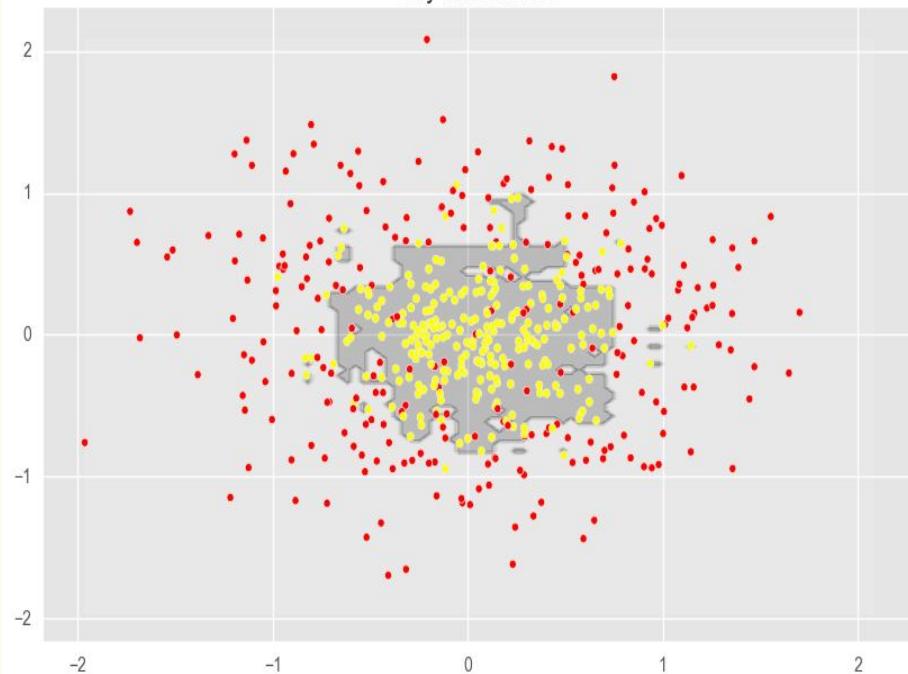
Случайный лес — это бэггинг над решающими деревьями, при обучении которых для каждого разбиения признаки выбираются из некоторого случайного подмножества признаков



Дерево решений



Случайный лес



3.2.4.3.1. `sklearn.ensemble.RandomForestClassifier`

```
class sklearn.ensemble. RandomForestClassifier(n_estimators=10, criterion='gini', max_depth=None,
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto',
max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False,
n_jobs=1, random_state=None, verbose=0, warm_start=False, class_weight=None)           [source]
```

A random forest classifier.

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if `bootstrap=True` (default).

Read more in the [User Guide](#).

Parameters: `n_estimators` : integer, optional (default=10)

The number of trees in the forest.

`criterion` : string, optional (default="gini")

The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. Note: this parameter is tree-specific.

`max_features` : int, float, string or None, optional (default="auto")

The number of features to consider when looking for the best split:

- If int, then consider `max_features` features at each split.
- If float, then `max_features` is a percentage and `int(max_features * n_features)` features are considered at each split.
- If "auto", then `max_features=sqrt(n_features)`.
- If "sqrt", then `max_features=sqrt(n_features)` (same as "auto").

3.2.4.3.2. `sklearn.ensemble.RandomForestRegressor`

```
class sklearn.ensemble. RandomForestRegressor (n_estimators=10, criterion='mse', max_depth=None,  
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto',  
max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False,  
n_jobs=1, random_state=None, verbose=0, warm_start=False) [source]
```

A random forest regressor.

A random forest is a meta estimator that fits a number of classifying decision trees on various sub-samples of the dataset and use averaging to improve the predictive accuracy and control over-fitting. The sub-sample size is always the same as the original input sample size but the samples are drawn with replacement if `bootstrap=True` (default).

Read more in the [User Guide](#).

Parameters: `n_estimators` : integer, optional (default=10)

The number of trees in the forest.

`criterion` : string, optional (default="mse")

The function to measure the quality of a split. Supported criteria are "mse" for the mean squared error, which is equal to variance reduction as feature selection criterion, and "mae" for the mean absolute error.

New in version 0.18: Mean Absolute Error (MAE) criterion.

`max_features` : int, float, string or None, optional (default="auto")

XGBoost

XGBoost — библиотека градиентного бустинга на деревьях решений с открытым исходным кодом.



LightGBM

LightGBM — библиотека градиентного бустинга на деревьях решений с открытым исходным кодом.



CatBoost

CatBoost — библиотека градиентного бустинга на деревьях решений с открытым исходным кодом.



<https://tech.yandex.ru/catboost/>

