«Машинное обучение»

# Контроль качества и выбор модели

Александр Дьяконов

# План

**Проблема выбора модели (в широком смысле)**

**Способы контроля**

**отложенный / скользящий / перекрёстный / бутстреп / по времени**

**Три золотых правила разбиения выборки**

**Моделируем реальность / нет утечкам / случайность**

**Где используется выбор CV-контроля**

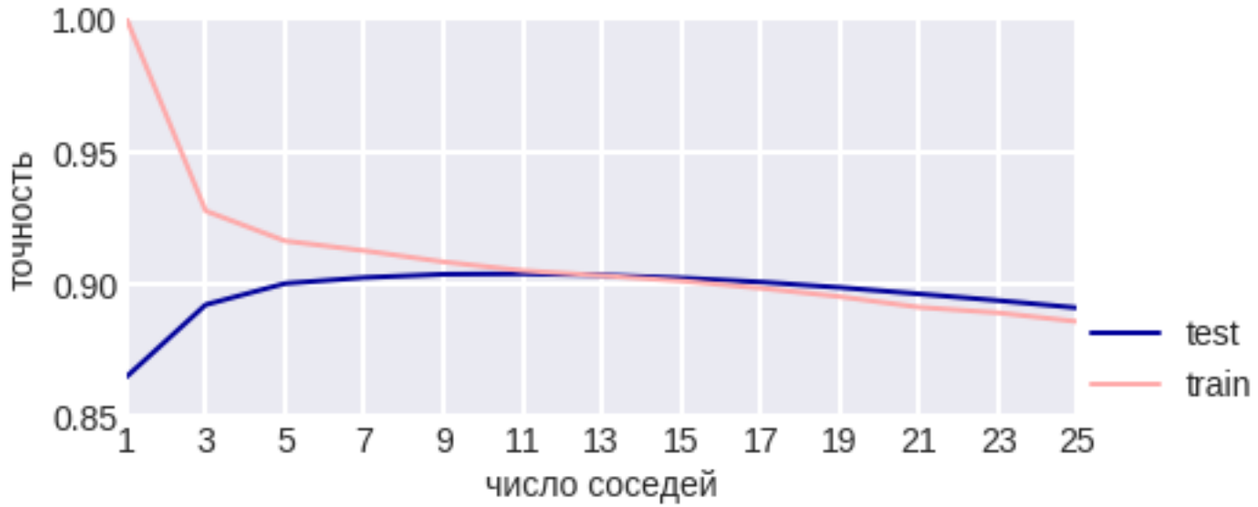
# Проблема контроля качества

**Ошибка на обучении (train error) и ошибка на контроле (test error) как правило очень различаются!**

**Как оценить качество / ошибку алгоритма?**

**model performance / model error**

**Нужен способ оценить качество работы (в будущем) алгоритма**



# Контроль качества: базовая идея

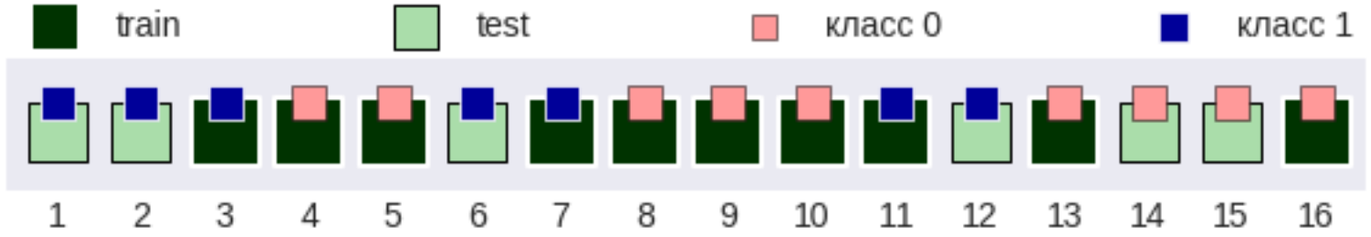
**Разбить выборку на две части: обучающую и тестовую (контрольную)**

# Отложенный контроль (held-out / validation data)

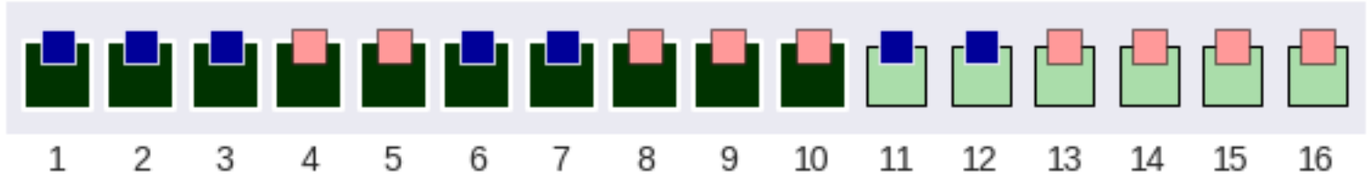
**Выборку делим на две части:**

- **обучение** – здесь обучение алгоритма
- **отложенный контроль** – здесь оценка качества / выбор алгоритма с наименьшей ошибкой

```
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, random_state=41)
```




```
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.33, shuffle=False)
```



# Отложенный контроль (held-out / validation data)

**+ обычно делят 80% / 20%**

больше тест – надёжнее оценка, больше обучение – алгоритм похож на финальный

**– оценка ошибки зависит от конкретной выбранной отложенной выборки,**

часто сильно меняется при другом выборе

**– если переобучить алгоритм для всех данных, то мы не знаем оценку его ошибки**

(в каком-то смысле, неустранимый недостаток)

# Способы контроля

- **отложенный контроль (held-out data, hold-out set)**
- **скользящий контроль (cross-validation)** – иногда так называют «всё»
- **бутстреп (bootstrap)**
- **контроль по времени (out-of-time-контроль)**

**Дальше подробно расскажем про разные способы контроля…**

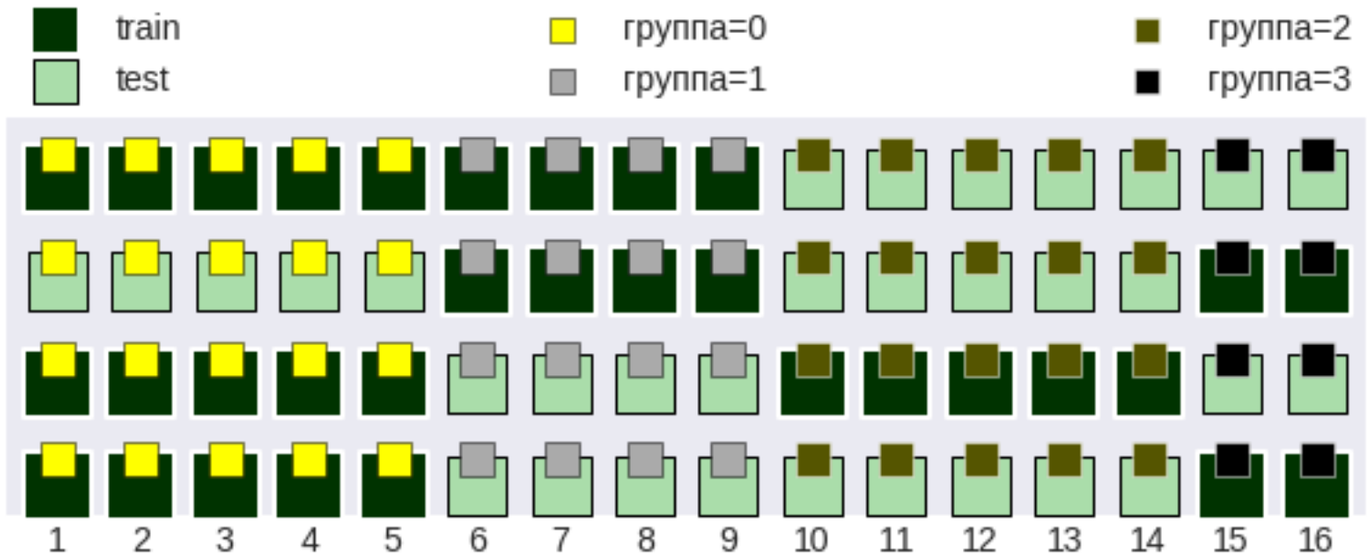
# Random Subsampling Cross-Validation

***k* раз случайно выбираем отложенный контроль,**
**усредняем ошибки на всех отложенных выборках**

```
sklearn.model_selection.ShuffleSplit(n_splits=4,
                                      test_size=0.3,
                                      train_size=None,
                                      random_state=None)
```

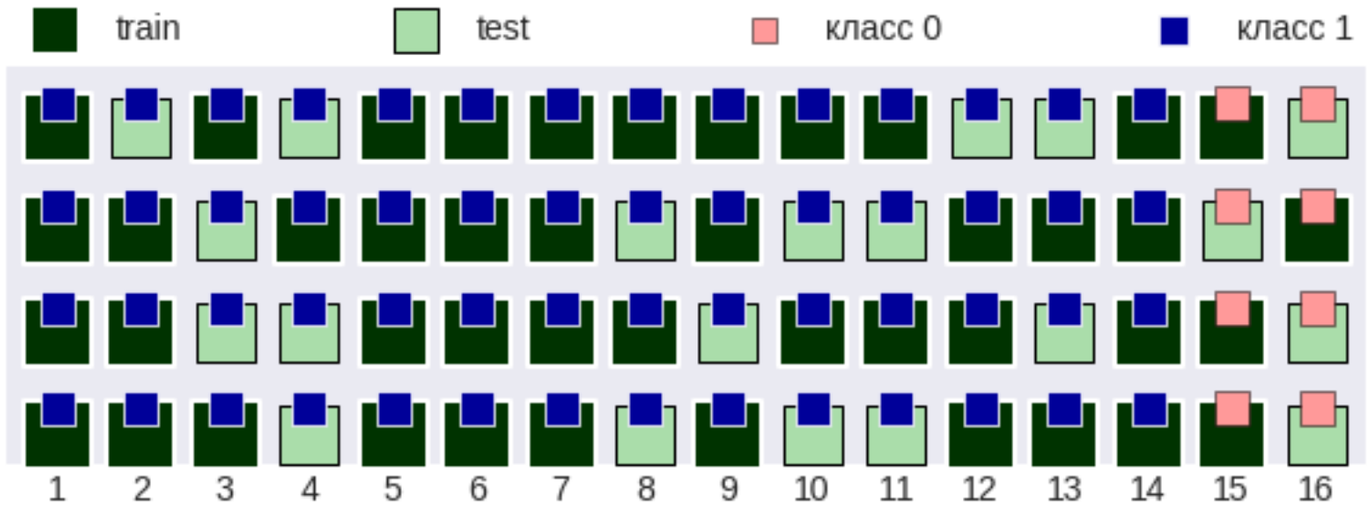
# Random Subsampling Cross-Validation: без разбиения групп

```
sklearn.model_selection.GroupShuffleSplit(n_splits=4,
                                          test_size=0.3,
                                          train_size=None,
                                          random_state=None)

for t, (itrain, itest) in enumerate(cv.split(x, groups=g)):
    ...
```



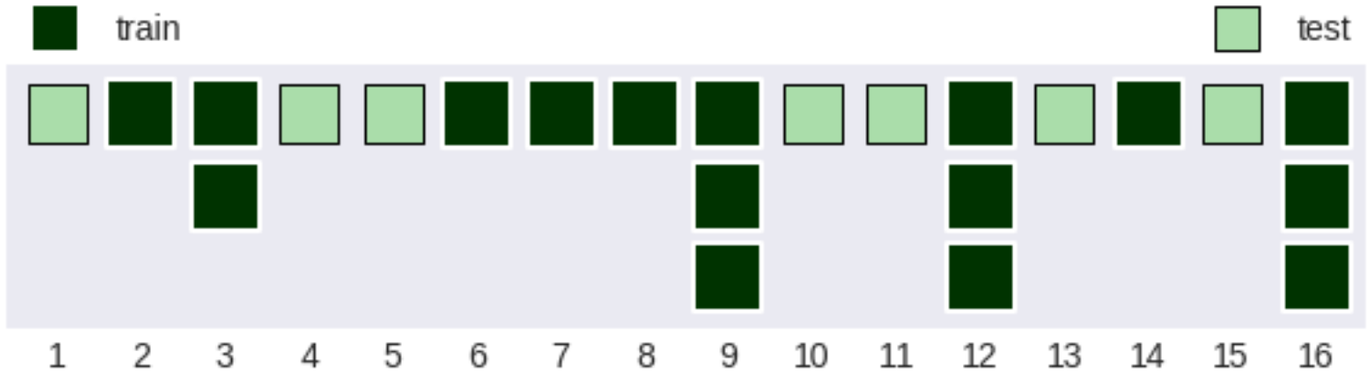
# Random Subsampling Cross-Validation: сохраняя пропорции классов

```
sklearn.model_selection.StratifiedShuffleSplit(n_splits=4,
                                                test_size=0.3,
                                                train_size=None,
                                                random_state=None)

for t, (itrain, itest) in enumerate(cv.split(x, groups=y)):
    ...
```



# Бутстреп (Bootstrap)

**с помощью выбора с возвращением**
**формируется подвыборка полного объёма *m*,**
**на которой производится обучение модели**
**на остальных объектах (которые не попали в обучение) – контроль**




```
i_train = [9, 16, 14, 9, 7, 12, 3, 12, 9, 8, 3, 2, 16, 12, 6, 16]
i_test =  [1, 4, 5, 10, 11, 13, 15]
```

# Бутстреп (Bootstrap)

**В контроль попадает примерно**

$$\left(1-\frac{1}{m}\right)^m \approx e^{-1} \approx 0.37 = 37\%$$ **выборки.**

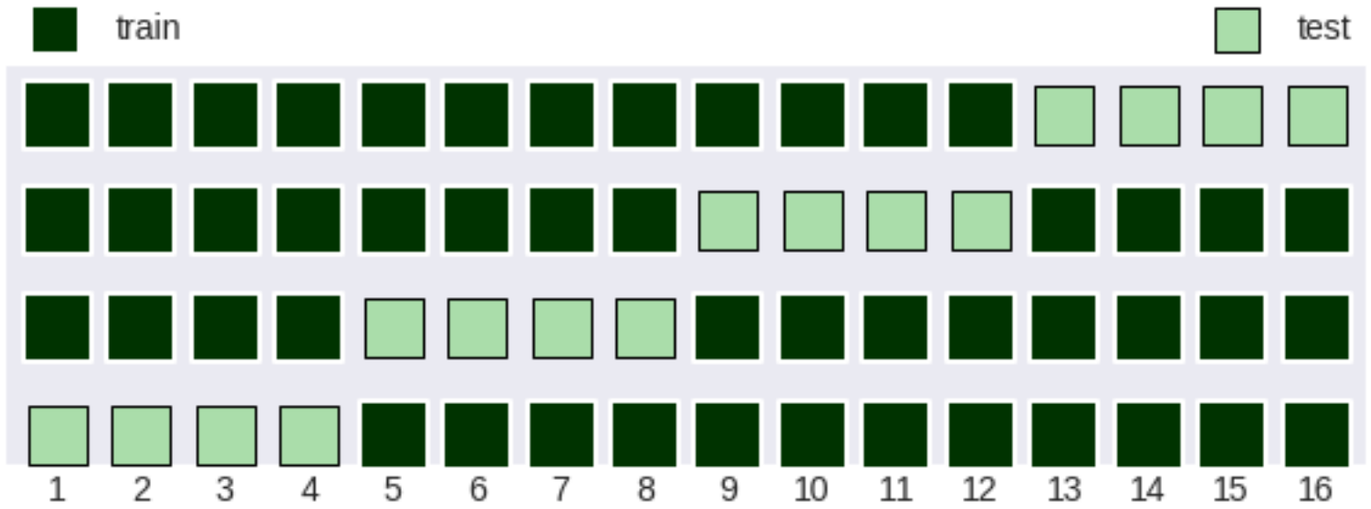
**+ модель учится на выборке того же объёма, что и итоговая**
**(которую мы обучим по всей выборке)**
**– использует не все данные**
**– есть дубликаты**

**+ с точки зрения распределения бутстреп-выборка похожа на исходную**

# Перекрёстная проверка по фолдам (k-fold cross-validation)

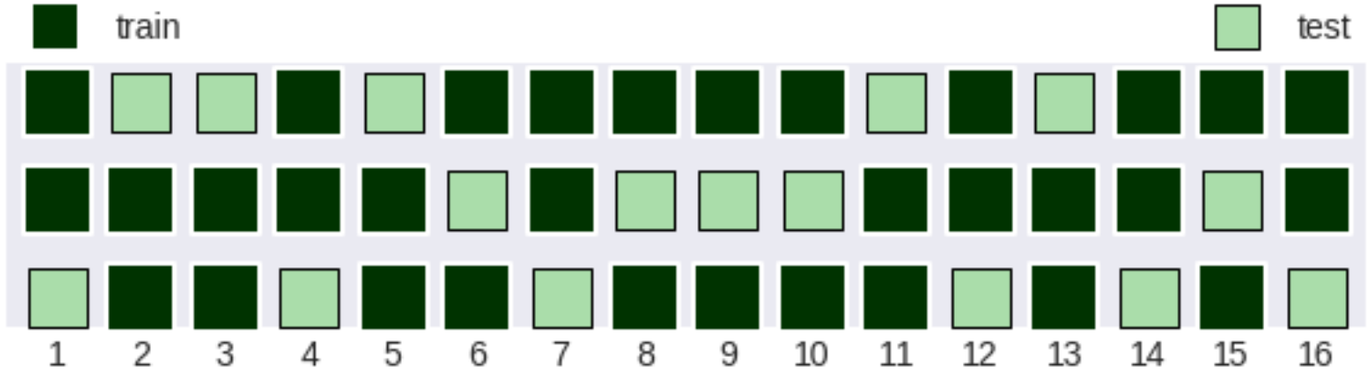
- **Разделить выборку на *k* примерно равных частей (обычно k=10)**
- **цикл по *i* = 1…*k***
  - ***i*-я часть – для теста, объединение остальных – для обучения**
- **усреднить *k* ошибок, вычисленных на разных итерациях цикла на валидациях (можно использовать дисперсию для оценки доверия к полученному качеству)**

```
sklearn.model_selection.KFold(n_splits='warn',
                              shuffle=False,
                              random_state=None)
```



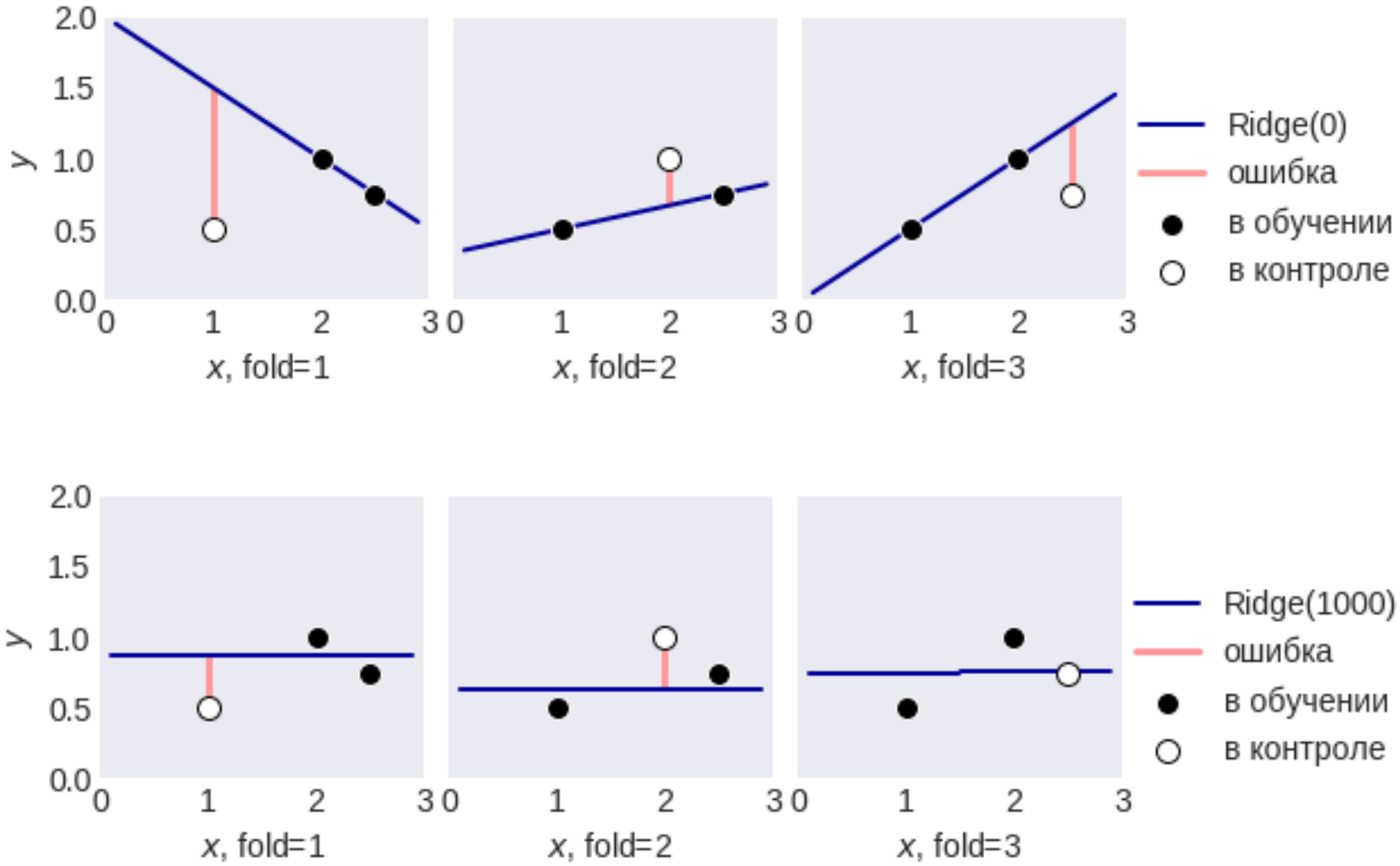
# Перекрёстная проверка по фолдам (k-fold cross-validation): с перемешиванием

```
sklearn.model_selection.KFold(n_splits=3, shuffle=True, random_state=None)
```




**k-fold CV = k-fold cross-validation**

# Перекрёстная проверка по фолдам: иллюстрация



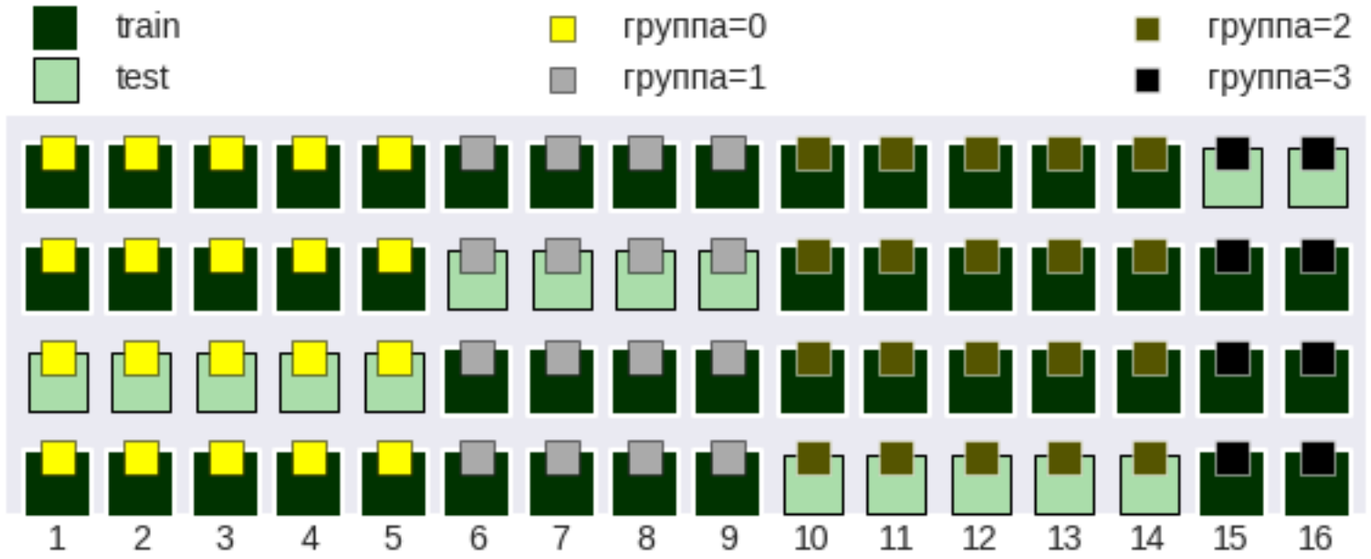
# Перекрёстная проверка по фолдам: сохранение пропорций классов

```
sklearn.model_selection.StratifiedKFold(n_splits='warn',
                                        shuffle=False,
                                        random_state=None)
```


**перемешиваем:** `shuffle=True`

# Перекрёстная проверка по фолдам: не разбиваем группы

```
sklearn.model_selection.GroupKFold(n_splits='warn')
```




**есть** `sklearn.model_selection.PredefinedSplit` **– разбиение индуцированное группами**

# Leave-one out cross-validation (LOOCV)

**$k$-fold при $k=m$**

`sklearn.model_selection.LeaveOneOut`


**показаны только первые 4 разбивки…**

**– может слишком долго вычисляться**

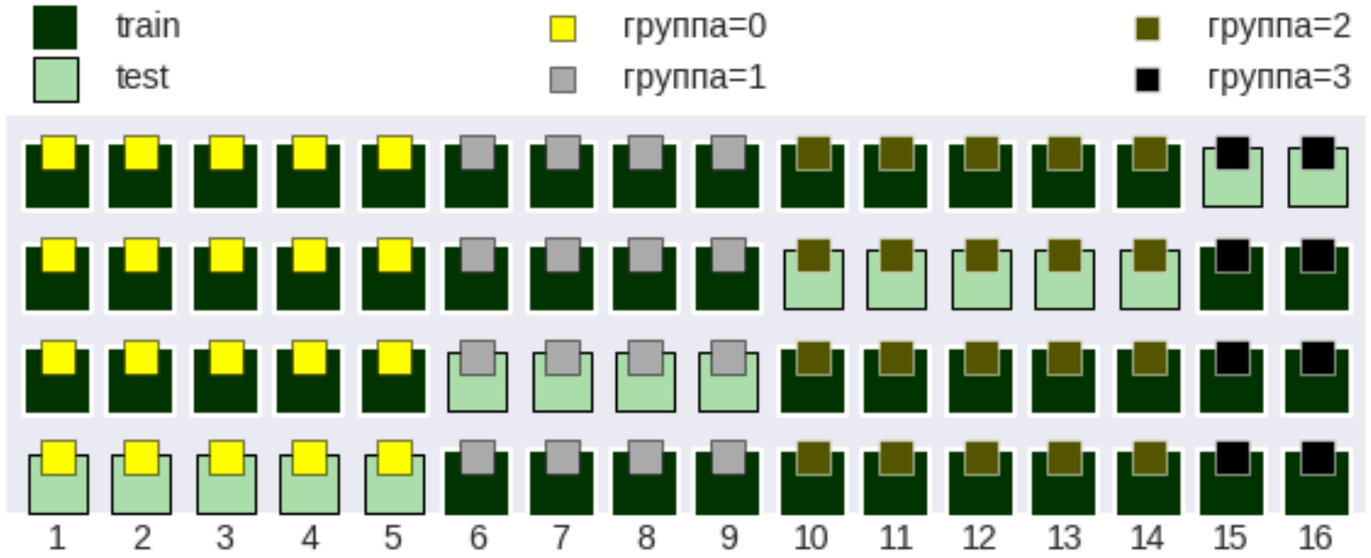
**+ хорошая для теории**

**ещё есть** `sklearn.model_selection.LeavePOut` **– всевозможные $P$-ки**

# Контроль по группам: LeaveOneGroupOut

## LeaveOneGroupOut: Контроль по одной группе

```
from sklearn.model_selection import LeaveOneGroupOut
```




**тонкость:**

**при оценки ошибки можно (нужно)**

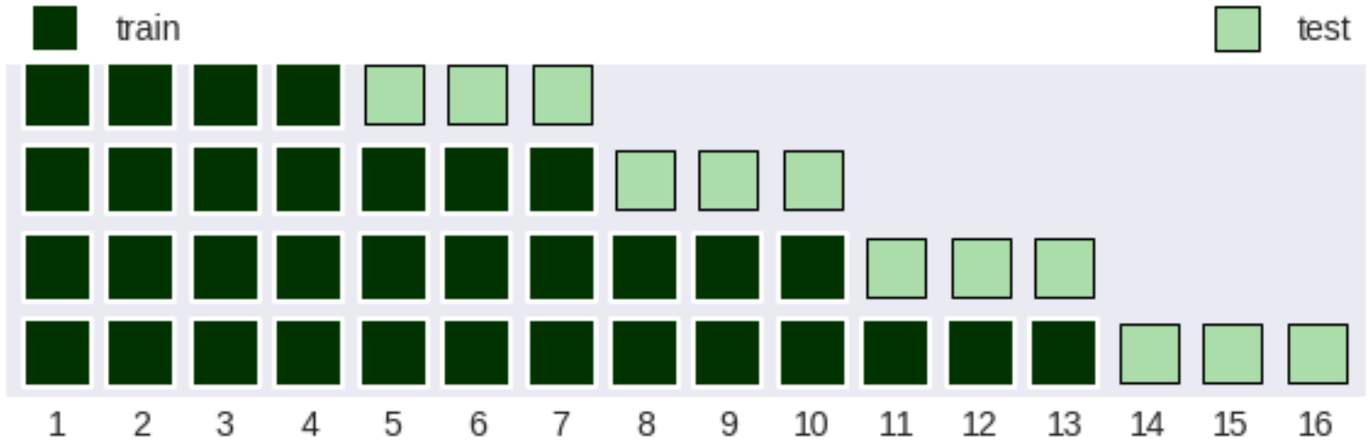
**учитывать мощность групп**

$$e = \sum_{t=1}^{k} \frac{m_t}{m} e_t$$

# Контроль по времени (Out-of-time-контроль)

## TimeSeriesSplit: разбиения временных рядов (Time series cross-validation)

```
sklearn.model_selection.TimeSeriesSplit(n_splits=5, # сколько делить
                                        max_train_size=None, # сколько делить
                                        test_size=None, # n_samples // (n_splits + 1)
                                                        # если gap=0
                                        gap=0) # сколько "откусывать" в конце
                                               # по умолчанию не используется
```




– часто не получится сделать много контролей (слишком маленькая предыстория)

\+ можно организовать «под контекст» (на следующий день, неделю, месяц)

# Схемы с повторениями: «сделать несколько раз»

```
model_selection.RepeatedKFold(n_splits=2, n_repeats=2, random_state=0)

model_selection.RepeatedStratifiedKFold(n_splits=2, n_repeats=2, random_state=0)
```

```
import numpy as np
from sklearn.model_selection import RepeatedStratifiedKFold
X = np.array([[1, 2], [3, 4], [1, 2], [3, 4]])
y = np.array([0, 0, 1, 1])
rskf = RepeatedStratifiedKFold(n_splits=2, n_repeats=2,
                               random_state=36851234)
for train_index, test_index in rskf.split(X, y):
    print("TRAIN:", train_index, "TEST:", test_index)
    X_train, X_test = X[train_index], X[test_index]
    y_train, y_test = y[train_index], y[test_index]
```

```
TRAIN: [1 2] TEST: [0 3]
TRAIN: [0 3] TEST: [1 2]
TRAIN: [1 3] TEST: [0 2]
TRAIN: [0 2] TEST: [1 3]
```

# Модификация основной идея для выбора и тестирования

**Обучающая выборка – Training Set**

обучение модели (настройка её параметров)

**Валидационная выборка – Validation Set**

выбор пайплайна ( модели / гиперпараметров / признаков)

иногда: локальный контроль

**Тестовая выборка – Test Set**

оценка качества алгоритма

иногда: итоговая оценка

## Минутка кода: оценка модели с помощью выбранного контроля

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import ShuffleSplit
from sklearn.linear_model import LogisticRegression

logreg = LogisticRegression()
cv = ShuffleSplit(n_splits=3, test_size=0.1,
                  train_size=None, random_state=None)
cross_val_score(logreg, X, y, cv=cv)
```

У этих функций много параметров…

Они (функции) «понимают» друг друга

Если не указываем скорер – используется встроенный (в модель)

**Есть аналогичная функция, которая сохраняет ещё время обучения и работы:**

`sklearn.model_selection.cross_validate`

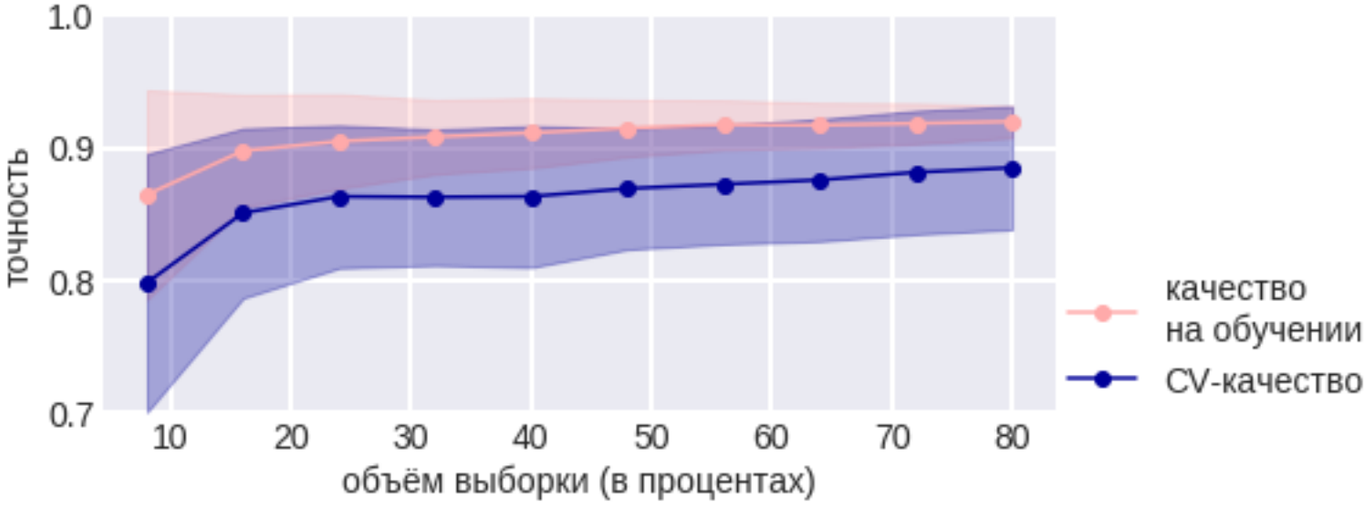
**Перестановочный тест для оценки неслучайности результата:**

`sklearn.model_selection.permutation_test_score`

# Кривые обучения (Learning Curves)

**Делим данные на обучение и контроль (м.б. очень много раз)**
**Обучаемся на k% от обучающей выборки для разных k**
**Строим графики ошибок/качества на train/CV от k**

```
model_selection.learning_curve
train_sizes, train_scores, test_scores = learning_curve(estimator, X, y, cv=cv,
                                                        n_jobs=n_jobs, train_sizes=train_sizes)
```



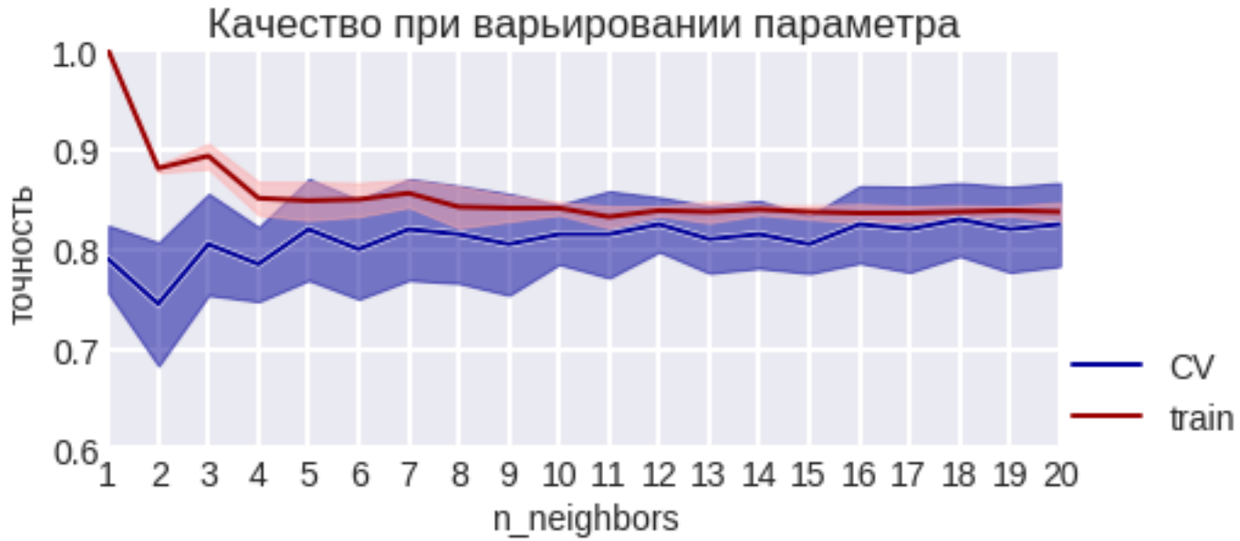

**Есть зазор между обучением и CV**

**Тонкость: 100% – вся выборка, но здесь** `test_size=0.2`

# Качество от параметров

**Валидационная кривая (Validation Curve)** показывает зависимость качества / ошибки при выбранной схеме контроля от значений гиперпараметров.

`sklearn.model_selection.validation_curve`



# Перебор значений гиперпараметров

**Делим данные на обучение и контроль (м.б. очень много раз)**

**При разных значениях параметров обучаемся и проверяем качество**

```
from sklearn.model_selection import GridSearchCV
parameters = {'metric':('euclidean', 'manhattan', 'chebyshev'),
              'n_neighbors':[1, 3, 5, 7, 9, 11] , scoring='roc_auc'}
clf = GridSearchCV(estimator, parameters, cv=5)
clf.fit(X, y)
clf.cv_results_['mean_test_score']
```

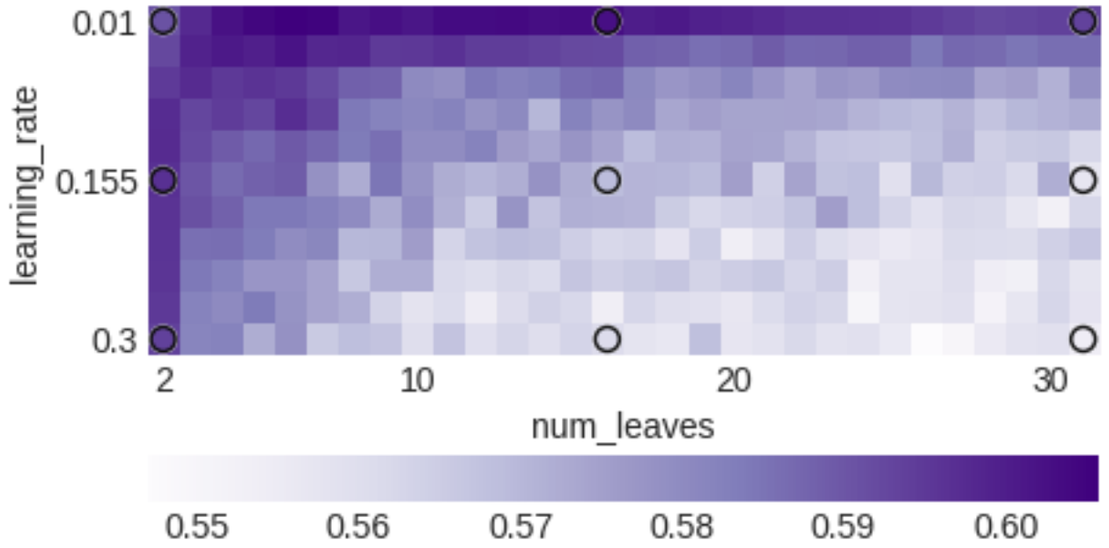
| | $k=1$ | $k=3$ | $k=5$ | $k=7$ | $k=9$ | $k=11$ |
|---|---|---|---|---|---|---|
| **euclidean** | 76.0 | 77.0 | 79.0 | 78.5 | 80.5 | 82.5 |
| **manhattan** | 74.0 | 74.0 | 79.0 | 79.5 | 80.5 | 81.0 |
| **chebyshev** | 76.5 | 78.5 | 80.0 | 80.0 | 81.0 | 81.5 |

**Есть также случайный поиск** `model_selection.RandomizedSearchCV`

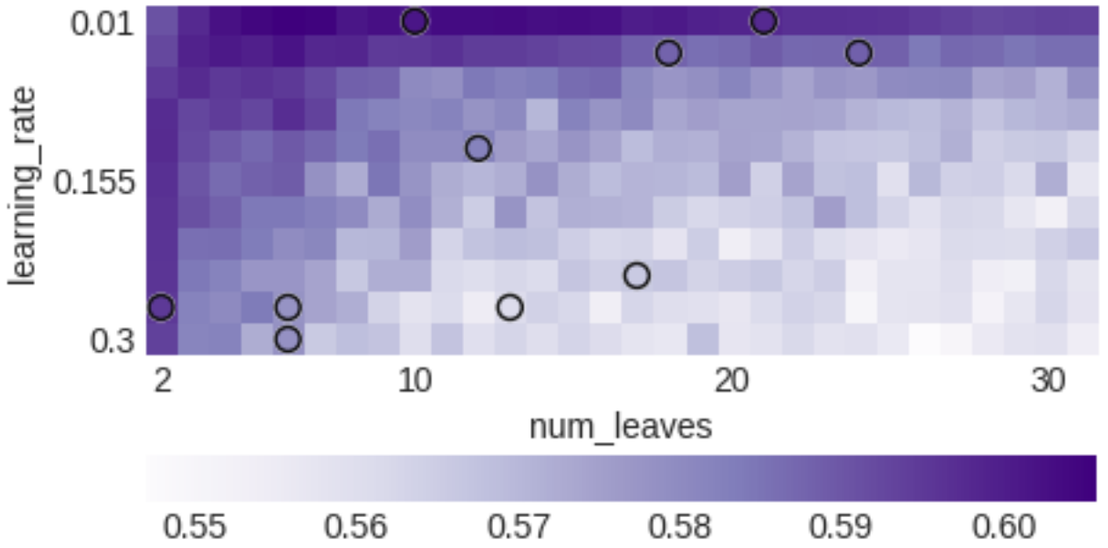
**(тут есть «число итераций», можно передавать распределения параметров)**

# Перебор параметров: случайный поиск считают предпочтительным

GridSearchCV




RandomizedSearchCV




```
import lightgbm as lgb
model = lgb.LGBMClassifier(n_estimators=100, subsample=0.75, colsample_bytree=0.75)

from sklearn.model_selection import GridSearchCV
parameters = {'num_leaves': np.arange(2, 32),
              'learning_rate':np.linspace(0.01, 0.3, 11)}
clf = GridSearchCV(model, parameters, cv=5, scoring='roc_auc')
clf.fit(X, y)
```

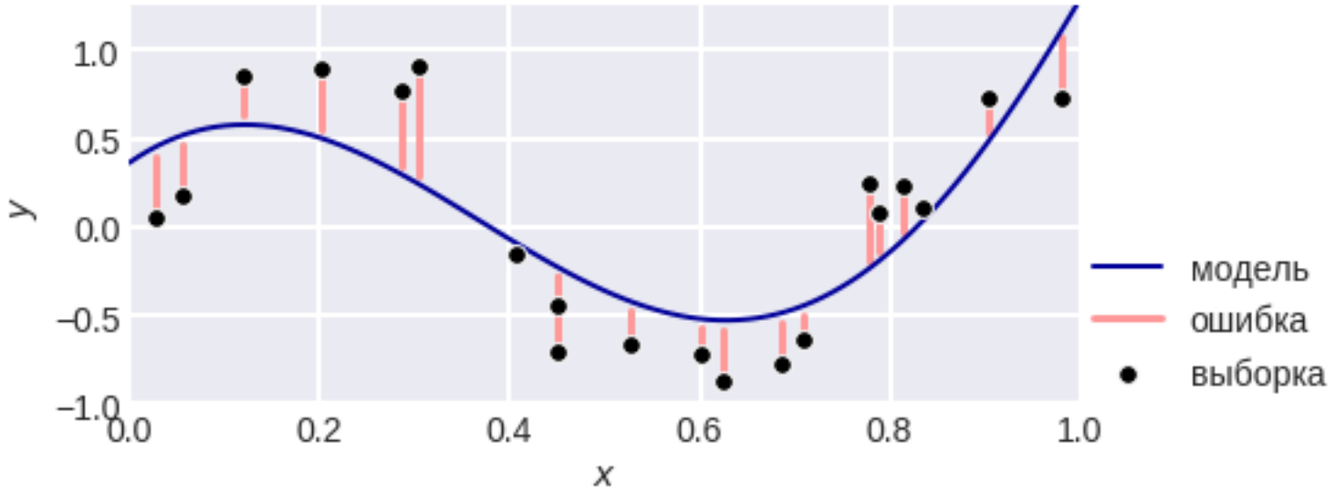
# Итог

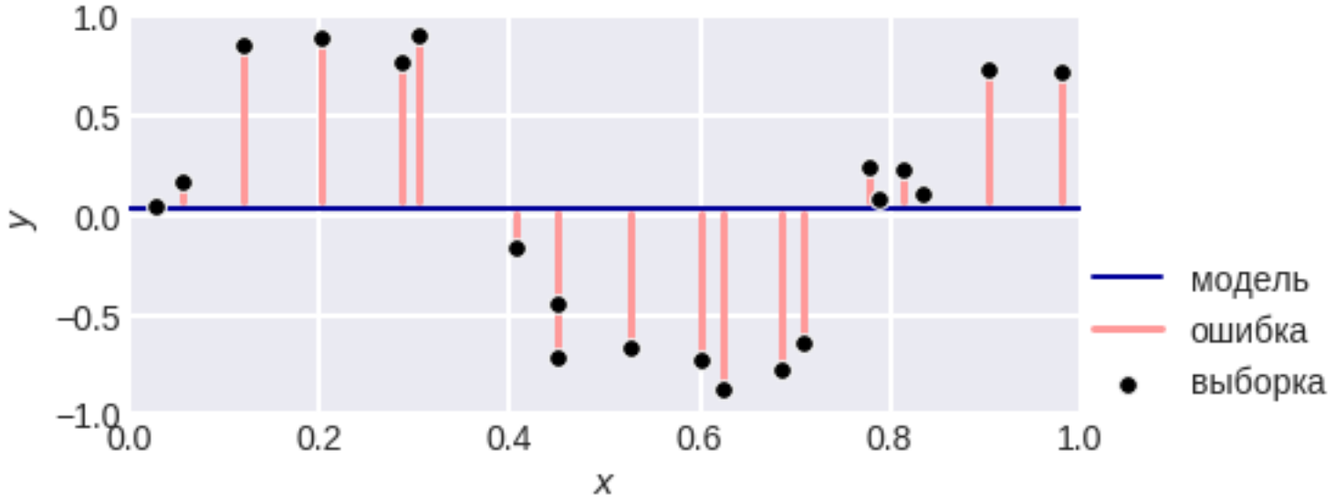
**Правильная организация контроля – важная часть решения задачи**

**Кривые качества**

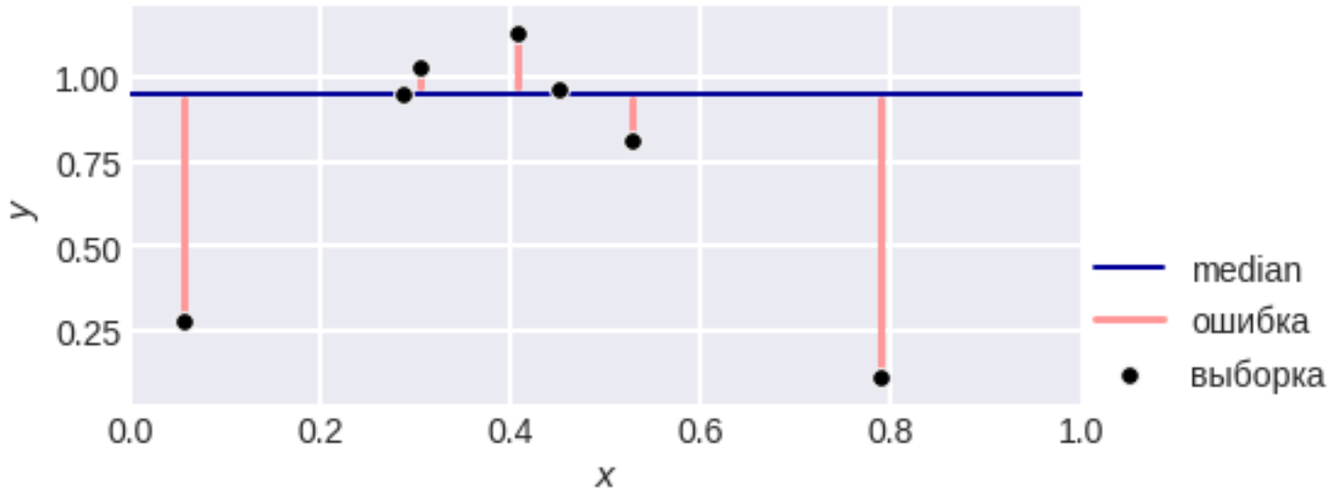
- validation curves (от значений)
- learning curves (от объёма выборки)

# Задача регрессии

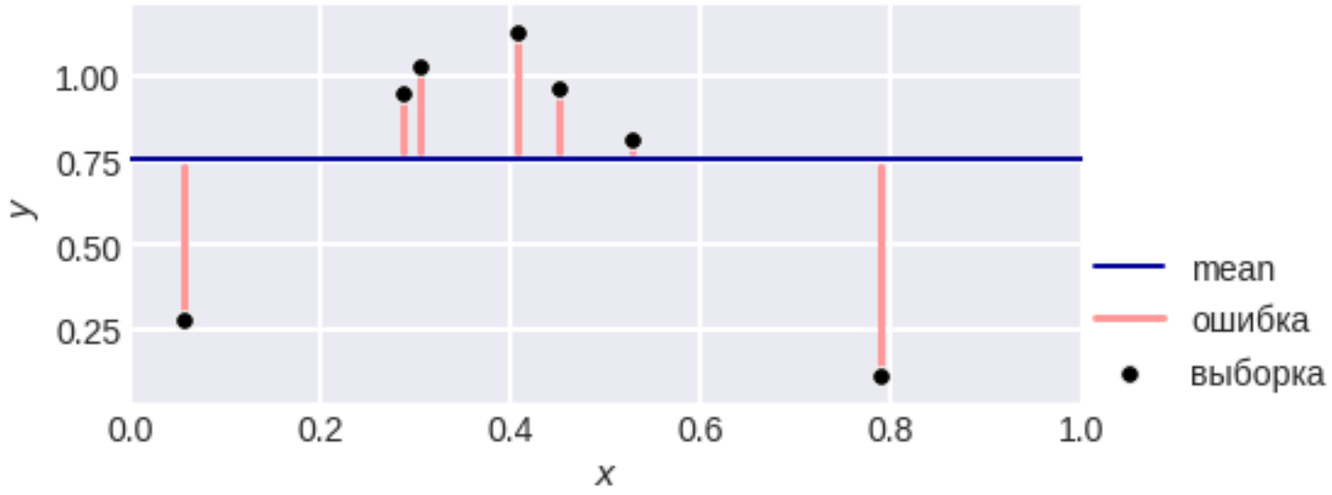


# Средний модуль отклонения – Mean Absolute Error (MAE), Mean Absolute Deviation (MAD)




$$\mathrm{MAE} = \frac{1}{m}\sum_{i=1}^{m} | a_i - y_i |$$




$$\mathrm{MSE} = \frac{1}{m}\sum_{i=1}^{m} | a_i - y_i |^2$$

# Root Mean Squared Error (RMSE) / Root Mean Square Deviation (RMSD)

$$\text{RMSE} = \sqrt{\frac{1}{m}\sum_{i=1}^{m} | a_i - y_i |^2}$$

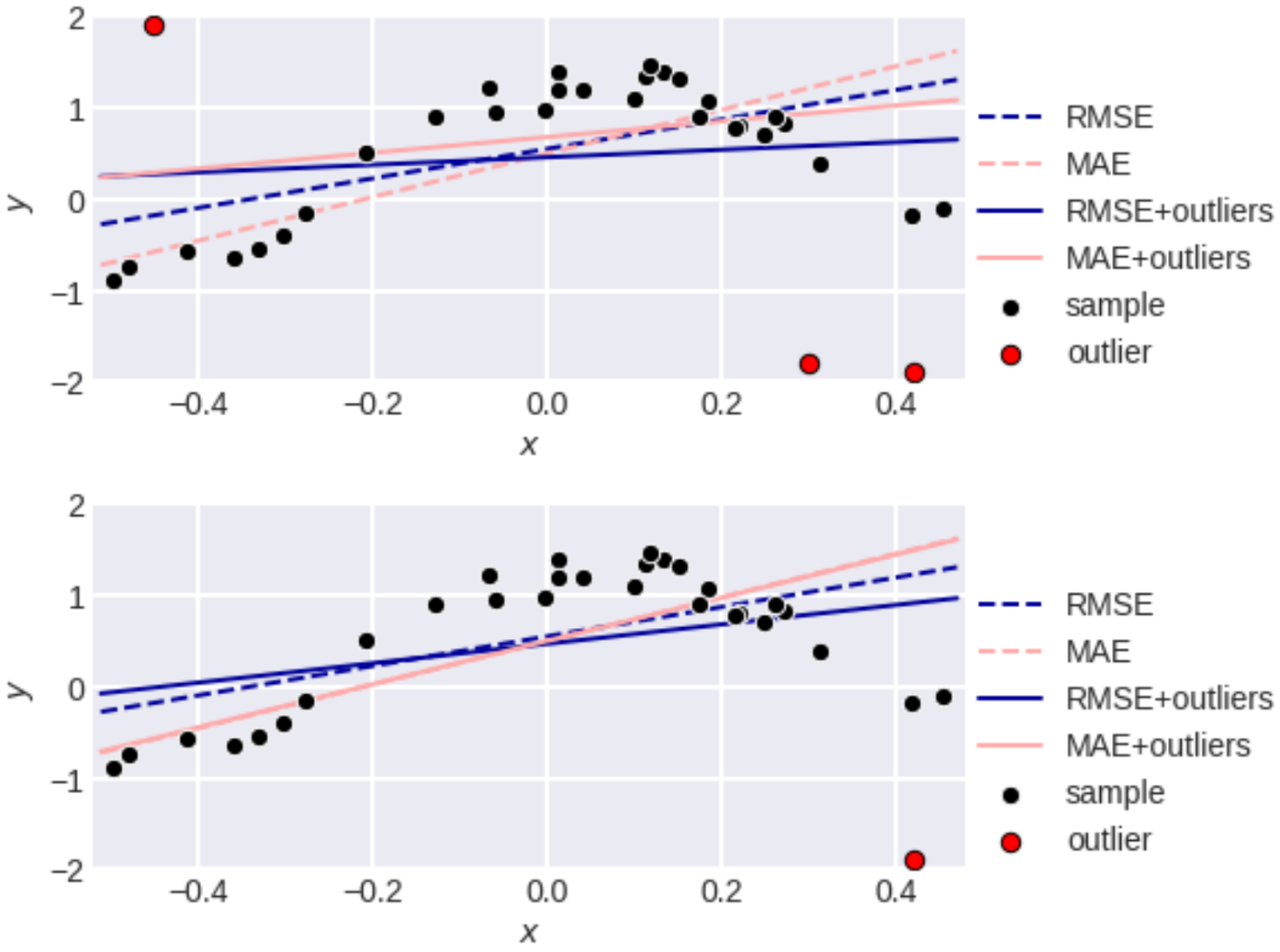
## Нормированная версия: коэффициент детерминации $R^2$ (Coefficient of Determination)

$$R^2 = 1 - \frac{\sum_{i=1}^{m} | a_i - y_i |^2}{\sum_{i=1}^{m} | \bar{y} - y_i |^2}$$

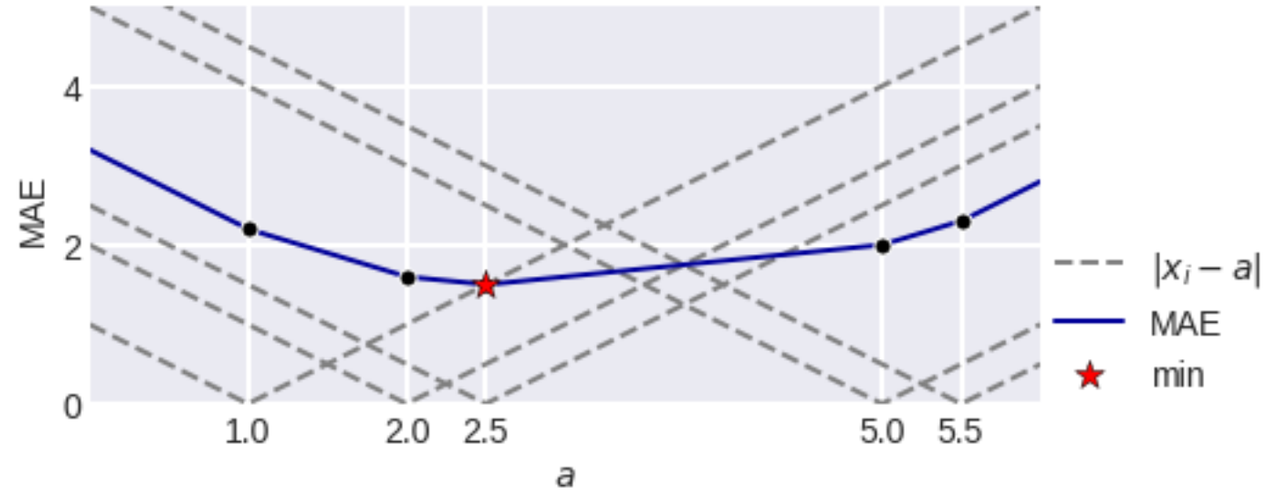
$$\bar{y} = \frac{1}{m}\sum_{i=1}^{m} y_i$$

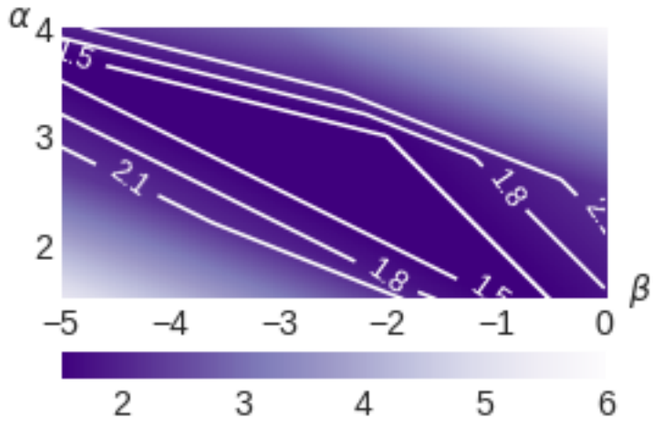
# Различия MSE и MAE

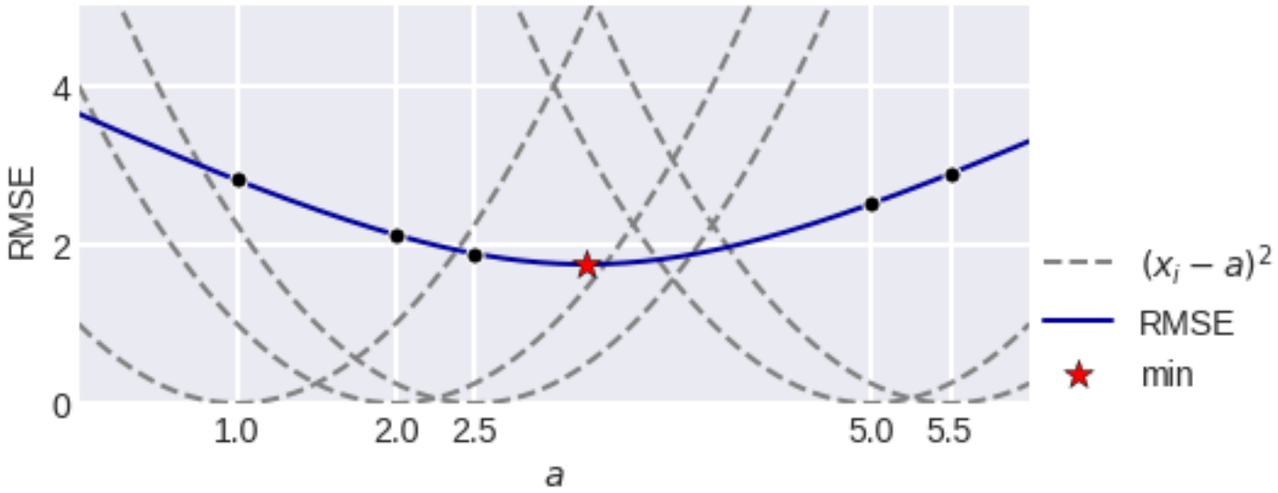
## Устойчивость к выбросам...

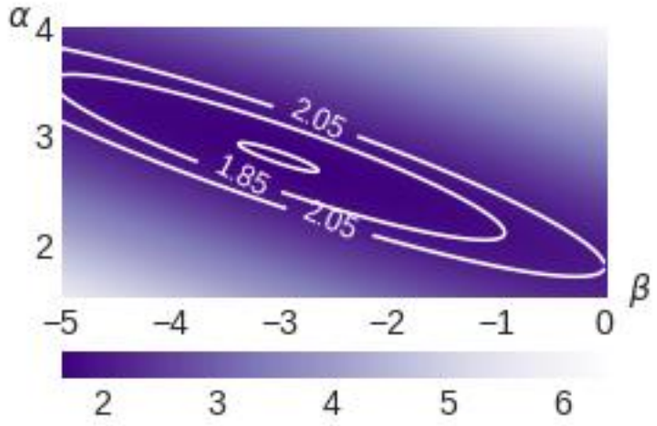


# Минимизируемая функция



# Symmetric mean absolute percentage error (SMAPE or sMAPE)

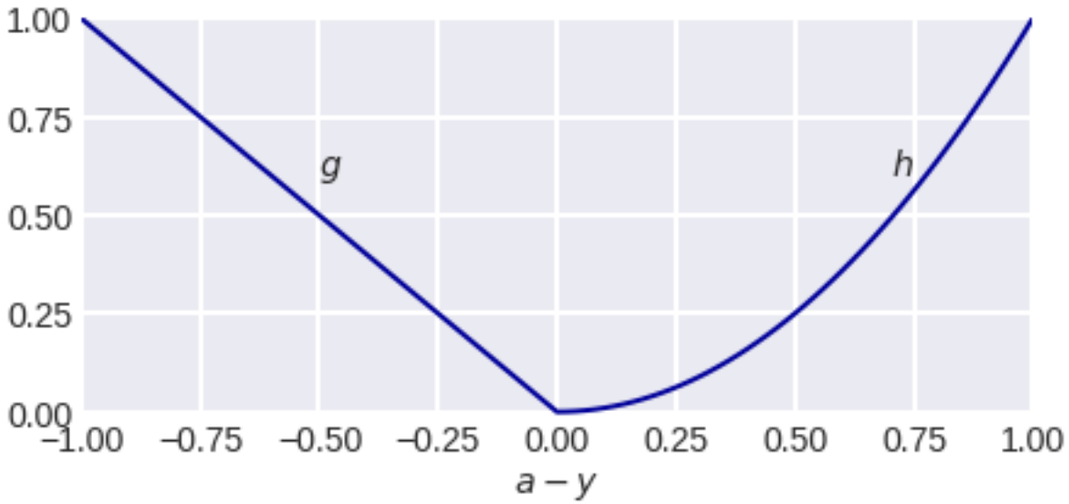
$$\text{SMAPE} = \frac{2}{m}\sum_{i=1}^{m}\frac{|\,y_i - a_i\,|}{y_i + a_i} = 100\% \cdot \frac{1}{m}\sum_{i=1}^{m}\frac{|\,y_i - a_i\,|}{(y_i + a_i)/2}$$

$$\text{MAPE} = \frac{1}{m}\sum_{i=1}^{m}\frac{|\,y_i - a_i\,|}{|\,y_i\,|}$$

$$\text{PMAD} = \frac{\sum_{i=1}^{m}|\,y_i - a_i\,|}{\sum_{i=1}^{m}|\,y_i\,|}$$

# Несимметричные функции потерь

$$\frac{1}{m}\sum_{i=1}^{m}\begin{cases} g(|\,y_i - a_i\,|), & y_i < a_i, \\ h(|\,y_i - a_i\,|), & y_i \geq a_i, \end{cases}$$




**Зачем нужны такие функции?**

# Метрики в регрессии: минутка кода

```
from sklearn.metrics import r2_score
from sklearn.metrics import mean_absolute_error
from sklearn.metrics import mean_squared_error
from sklearn.metrics import mean_squared_log_error
from sklearn.metrics import median_absolute_error
from sklearn.metrics import explained_variance_score
# R^2
print (r2_score(y, a),
       1 - np.mean((y - a) ** 2) / np.mean((y - np.mean(y)) ** 2))
# MAE
print (mean_absolute_error(y, a),
       np.mean(np.abs(y - a)))
# MSE
print (mean_squared_error(y, a),
       np.mean((y - a) ** 2))
# MSLp1E
print (mean_squared_log_error(y, a),
       np.mean((np.log1p(y) - np.log1p(a)) ** 2))
# MedAE
print (median_absolute_error(y, a),
       np.median(np.abs(y - a)))
```

# Итоги в регрессии

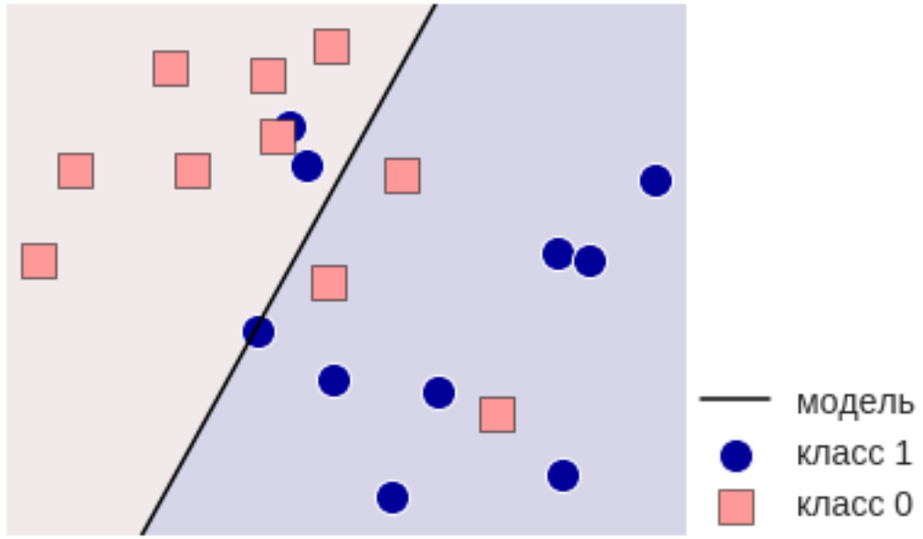
**средний модуль отклонения MAE (MAD)**

**средний квадрат отклонения MSE**

**Иногда попадаются обобщения MAE и RMSE**

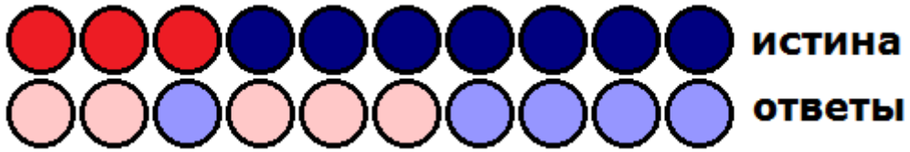
**Несимметричные ошибки**

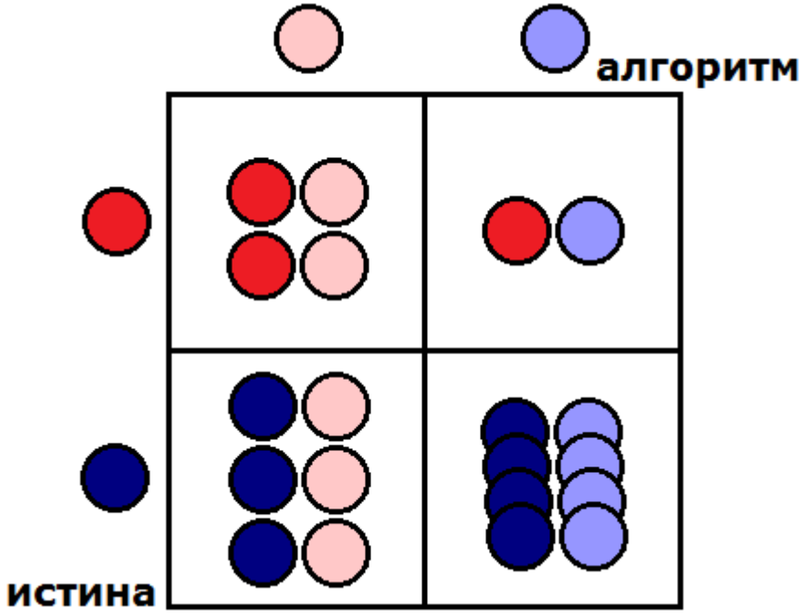
# Задача бинарной классификации




**сначала – чёткая классификация**

# «Confusion Matrix» в задаче классификации с двумя классами



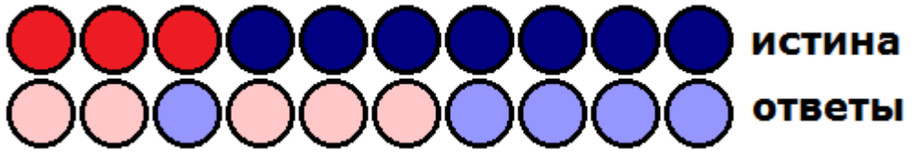

| | $a=0$ | $a=1$ |
|---|---|---|
| $y=0$ | 13599 | 2600 |
| $y=1$ | 898 | 903 |

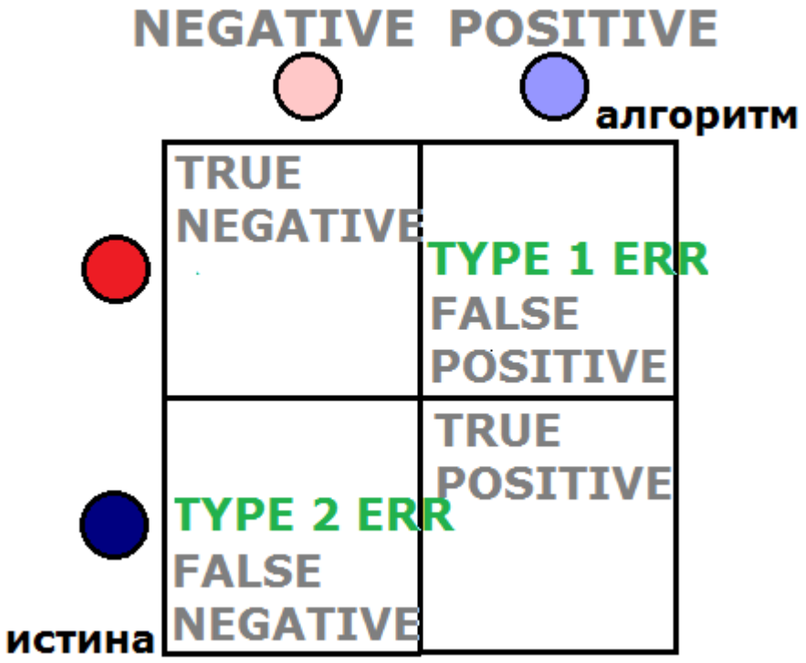
**в scikit-learn-е такая ориентация!**

**Иногда: наоборот!**

```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, a_test)
```

# Задача классификации с двумя классами




```
tn, fp, fn, tp = confusion_matrix(y, a).ravel() # вычисление tn, ...
```

# Как запомнить названия ошибок

**1 рода – не учил, но сдал (= знает по мнению экзаменатора)**

**2 рода – учил, но не сдал (= не знает по мнению экзаменатора)**


**Ошибка 1 рода**

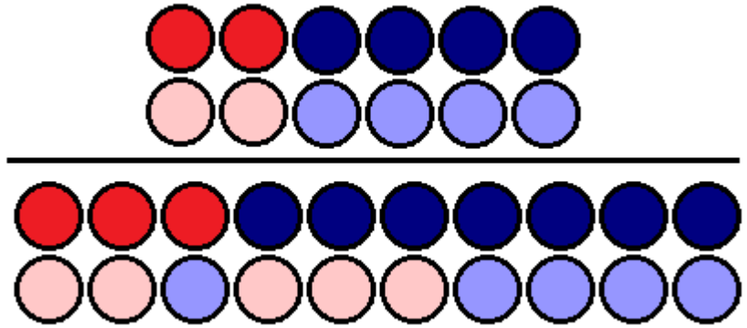

**Ошибка 2 рода**

FP/ $m$

FN/ $m$

# Точность Accuracy

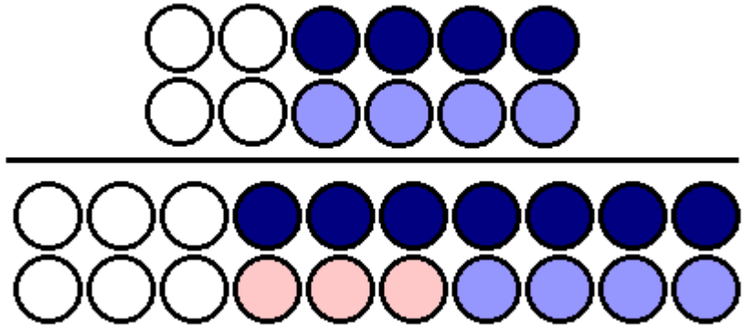
| | $a=0$ | $a=1$ |
|---|---|---|
| $y=0$ | 13599 | 2600 |
| $y=1$ | 898 | 903 |

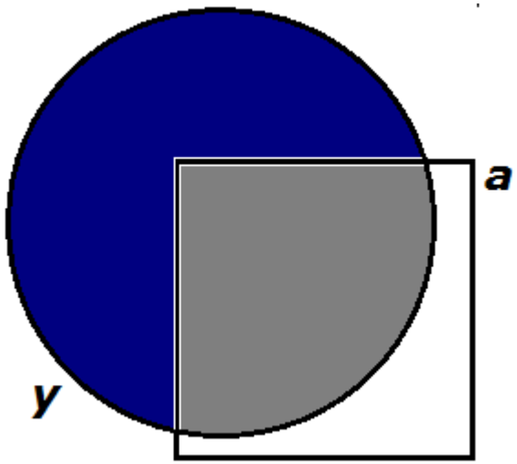


$$\text{Accuracy} = \frac{\text{TN}+\text{TP}}{\text{TN}+\text{FN}+\text{TP}+\text{FP}}$$

# Полнота (Sensitivity, True Positive Rate, Recall, Hit Rate)

| | $a=0$ | $a=1$ |
|---|---|---|
| $y=0$ | 13599 | 2600 |
| $y=1$ | 898 | 903 |



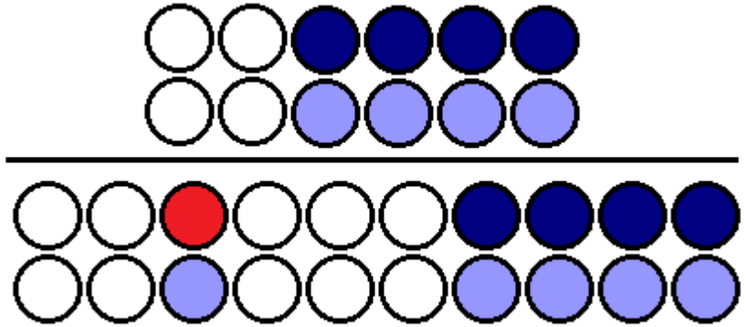


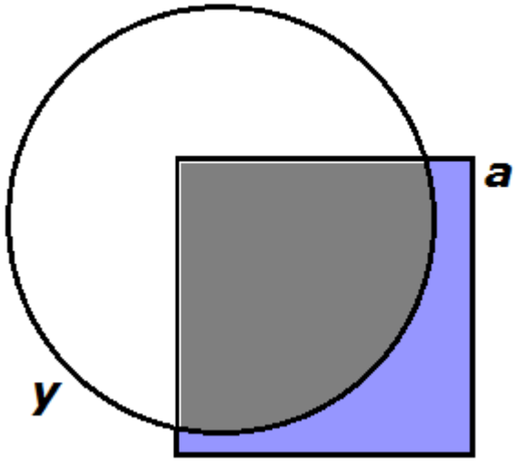

$$\text{TPR} = R = \frac{\text{TP}}{\text{TP}+\text{FN}}$$

**какой процент объектов положительного класса мы правильно классифицировали**

# Точность (Precision, Positive Predictive Value)

| | $a=0$ | $a=1$ |
|---|---|---|
| $y=0$ | 13599 | 2600 |
| $y=1$ | 898 | 903 |



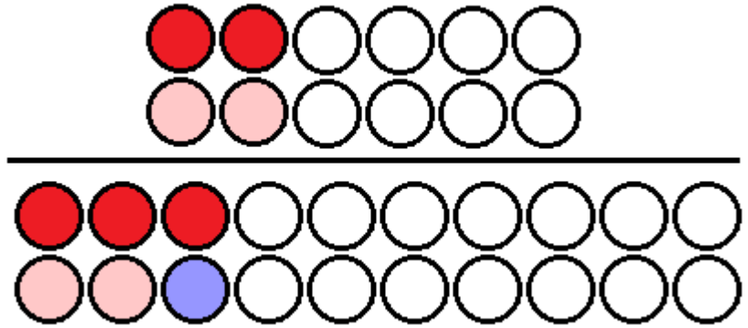


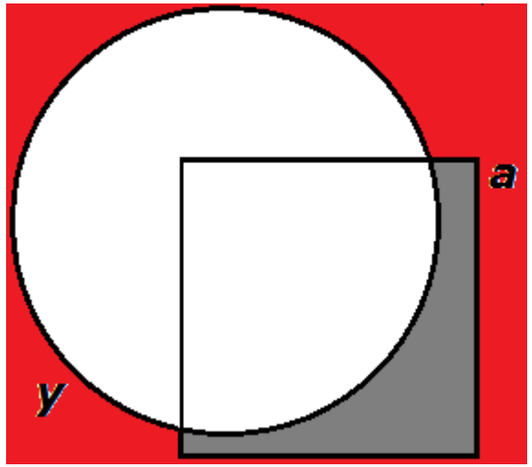

$$\text{PPV} = P = \frac{\text{TP}}{\text{TP}+\text{FP}}$$

**какой процент положительных объектов**
**(т.е. тех, что мы считаем положительными)**
**правильно классифицирован**

# Специфичность (Specificity, True Negative Rate)

| | $a=0$ | $a=1$ |
|---|---|---|
| $y=0$ | 13599 | 2600 |
| $y=1$ | 898 | 903 |





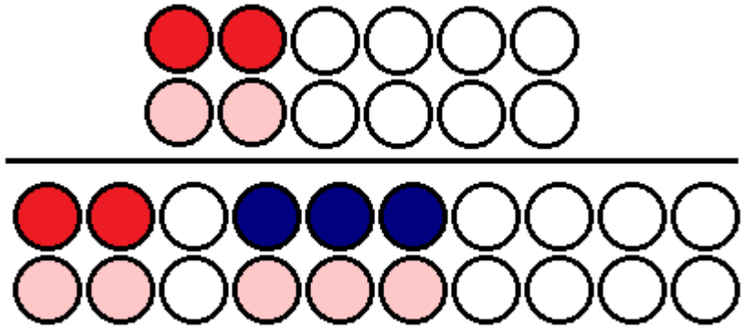

$$\text{TNR} = \text{Specificity} = R_0 = \frac{\text{TN}}{\text{TN}+\text{FP}}$$

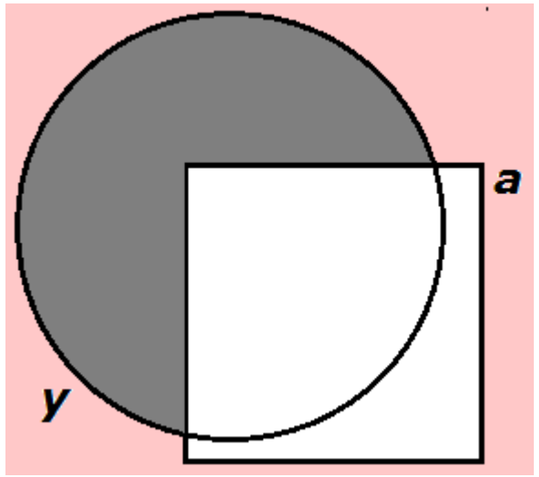
**процент правильно классифицированных объектов негативного класса**

**«полнота для негативного класса»!**

# Negative Predictive Value (Inverse Precision)

| | $a = 0$ | $a = 1$ |
|---|---|---|
| $y = 0$ | 13599 | 2600 |
| $y = 1$ | 898 | 903 |






$$\text{NPV} = P_0 = \frac{\text{TN}}{\text{TN+ FN}}$$

точность для нулевого класса

# False Positive Rate (FPR, fall-out, false alarm rate)

| | $a=0$ | $a=1$ |
|---|---|---|
| $y=0$ | 13599 | 2600 |
| $y=1$ | 898 | 903 |

$$\text{FPR} = \frac{\text{FP}}{\text{TN}+\text{FP}} = 1 - \text{TNR} = 1 - \text{Specificity}$$

**доля объектов негативного класса,**
**которых мы ошибочно отнесли к положительному**

## $F_1$ score

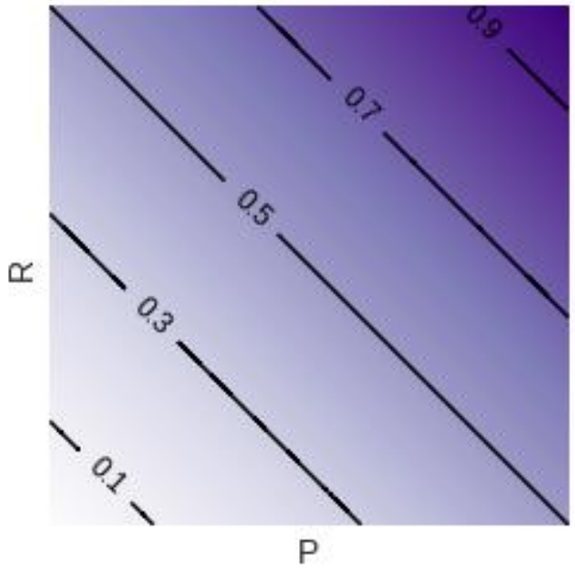
$$\frac{2}{\frac{1}{\mathrm{P}}+\frac{1}{\mathrm{R}}}=\frac{2}{\frac{1}{\mathrm{TP}/(\mathrm{TP}+\mathrm{FP})}+\frac{1}{\mathrm{TP}/(\mathrm{TP}+\mathrm{FN})}}=\frac{2\mathrm{TP}}{2\mathrm{TP}+\mathrm{FP}+\mathrm{FN}}$$

## $F_\beta$ score

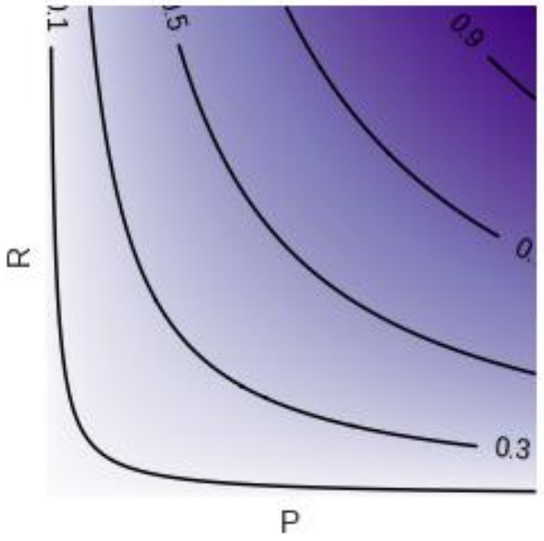
$$F_\beta=\frac{1}{\frac{\alpha}{\mathrm{P}}+\frac{1-\alpha}{\mathrm{R}}}=\frac{1}{\alpha}\frac{\mathrm{P}\cdot\mathrm{R}}{\mathrm{R}+\left(\frac{1}{\alpha}-1\right)\mathrm{P}}=(1+\beta^2)\frac{\mathrm{P}\cdot\mathrm{R}}{\mathrm{R}+\beta^2\mathrm{P}}$$

$$\beta^2=\left(\frac{1}{\alpha}-1\right)$$

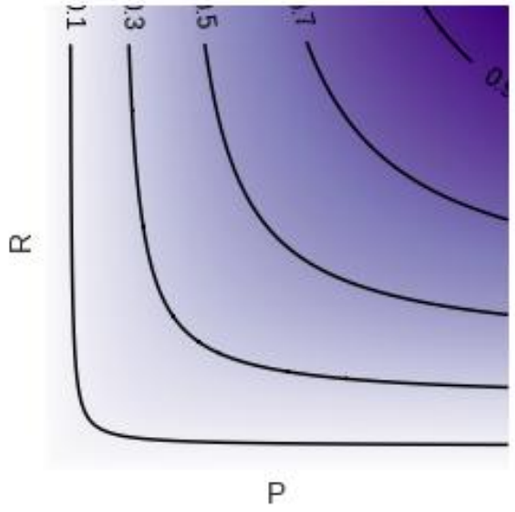
# Почему используется F-мера



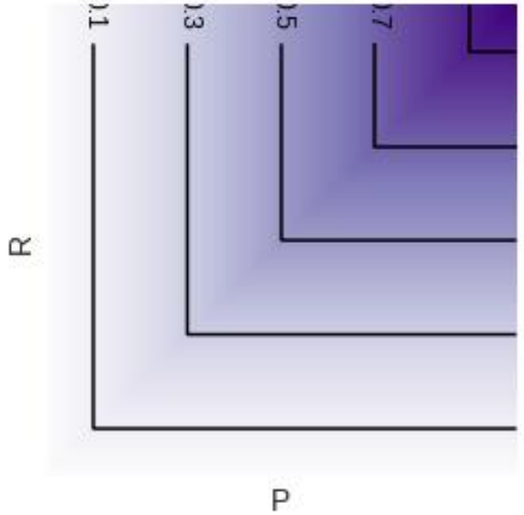

$(P+R)/2$




$\sqrt{P \cdot R}$

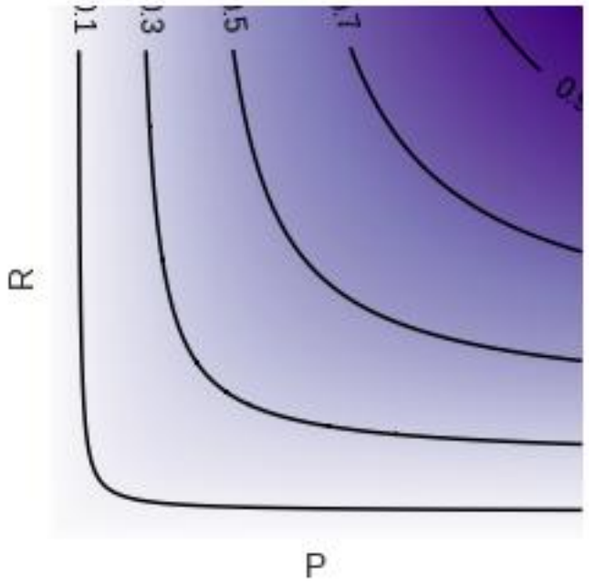



$2/(1/P+1/R)$

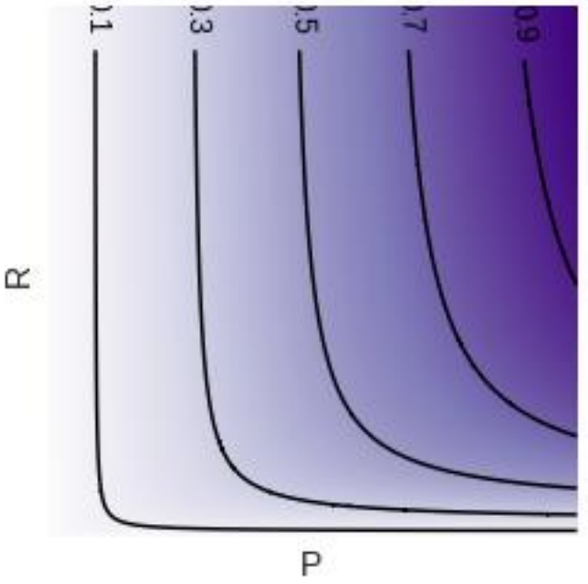



$\min(P,R)$

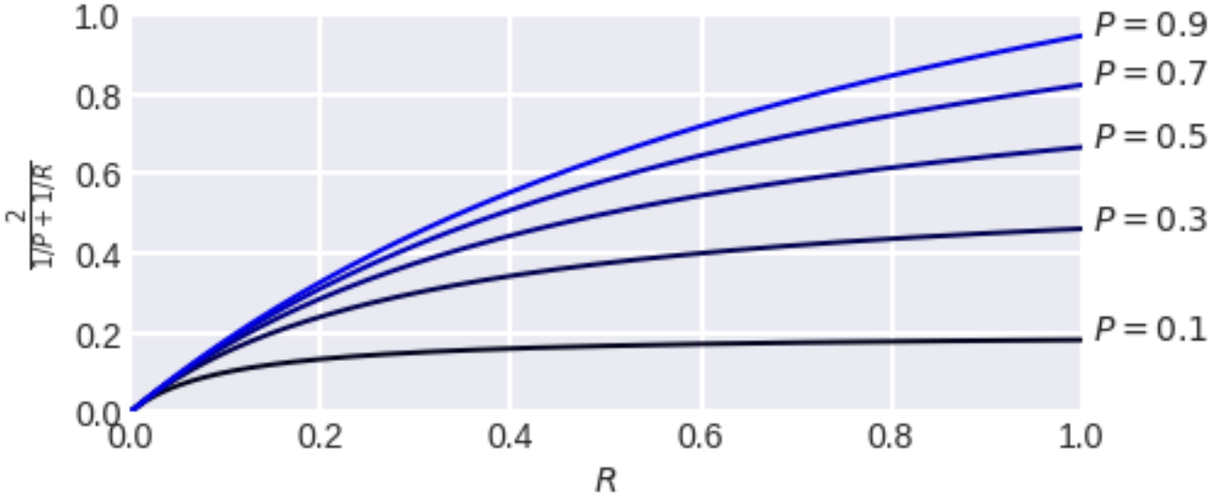
# Почему используется F-мера




$2/(1/P+1/R)$




$1/(0.9/P+0.1/R)$




**Можно сколь угодно улучшать один из показателей (R),**
**если второй не увеличивается (P), то качество ограничено**

# Минутка кода

```
from sklearn.metrics import classification_report
print (classification_report(y_test, a_test)) # нужен print
```

```
              precision    recall  f1-score   support

         0.0       0.94      0.84      0.89     16199
         1.0       0.26      0.50      0.34      1801

   micro avg       0.81      0.81      0.81     18000
   macro avg       0.60      0.67      0.61     18000
weighted avg       0.87      0.81      0.83     18000
```

```
from sklearn.metrics import cohen_kappa_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import matthews_corrcoef
from sklearn.metrics import f1_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import balanced_accuracy_score
```

| | score |
|---|---|
| cohen_kappa_score | 0.24 |
| accuracy_score | 0.81 |
| matthews_corrcoef | 0.26 |
| f1_score | 0.34 |
| roc_auc_score | 0.67 |
| balanced_accuracy_score | 0.67 |

# Итог по классификации

**В задаче чёткой бинарной классификации вся информация об ошибках в 2×2-матрице несоответствий**

**Много разных функционалов качества**

- **естественные**
- **из информационного поиска**
- **для учёта дисбаланса**

# Задача нечёткой бинарной классификации

**Теперь выдаём оценку принадлежности к классу 1**

$$y \in \{0, 1\}$$

$$a \in [0, 1]$$

**кроме меток {0, 1} возможны промежуточные значения**

# Log Loss

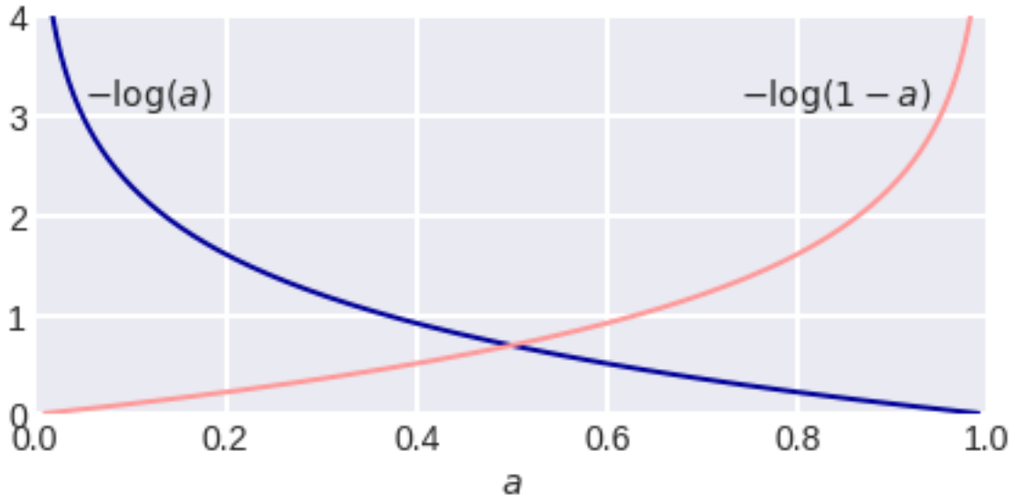
**В задаче классификации с двумя непересекающимися классами (0, 1), когда ответ – вероятность (?) принадлежности к классу 1**

$$\text{logloss} = -\frac{1}{m}\sum_{i=1}^{m}(y_i \log a_i + (1-y_i)\log(1-a_i))$$

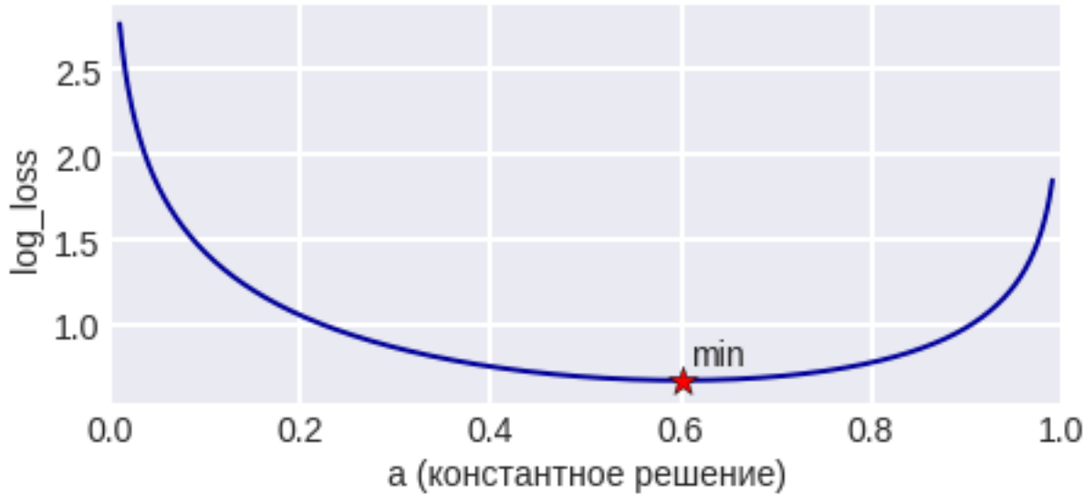
**Раздельная форма понятнее…**

$$-\begin{cases} \log a_i, & y_i = 1, \\ \log(1-a_i), & y_i = 0. \end{cases}$$

**Нельзя ошибаться!**



# Log Loss: Оптимальная константа для конечной выборки




$$-\frac{1}{m}\sum_{i=1}^{m}(y_i \log a + (1-y_i)\log(1-a)) \to \min_a$$

$$-\frac{m_1}{m}\log a - \frac{m_0}{m}\log(1-a) \to \min_a$$

$$a = \frac{m_1}{m}$$

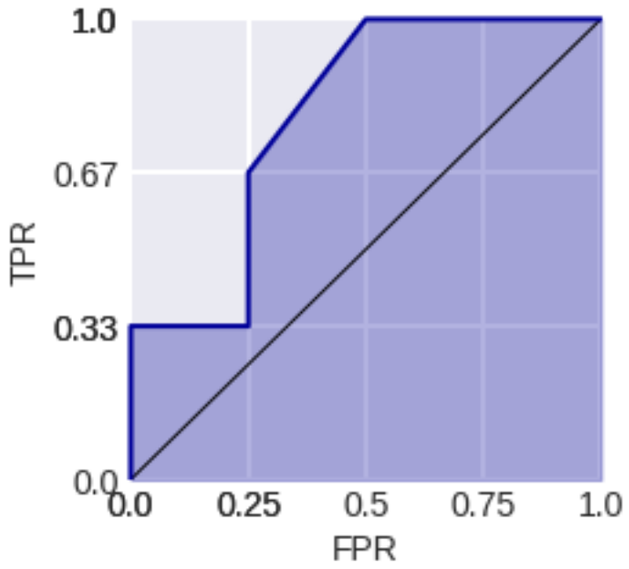
# ROC и AUC ROC

**ROC = receiver operating characteristic**

**Функционал зависит не от конкретных значений, а от их порядка**

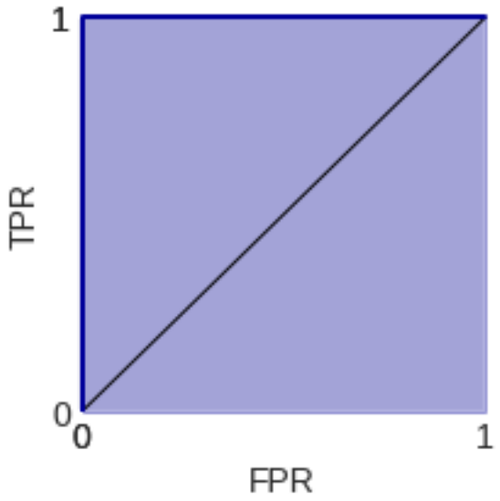
| | оценка | класс |
|---|---|---|
| **0** | 0.5 | 0 |
| **1** | 0.1 | 0 |
| **2** | 0.2 | 0 |
| **3** | 0.6 | 1 |
| **4** | 0.2 | 1 |
| **5** | 0.3 | 1 |
| **6** | 0.0 | 0 |

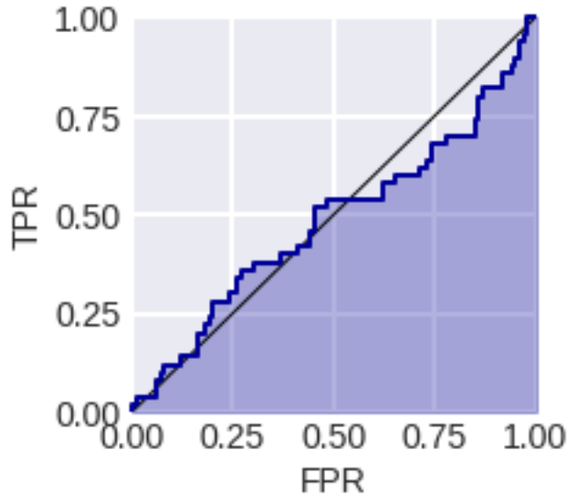
| | оценка | класс | ответ |
|---|---|---|---|
| **3** | 0.6 | 1 | 1 |
| **0** | 0.5 | 0 | 1 |
| **5** | 0.3 | 1 | 1 |
| **2** | 0.2 | 0 | 0 |
| **4** | 0.2 | 1 | 0 |
| **1** | 0.1 | 0 | 0 |
| **6** | 0.0 | 0 | 0 |

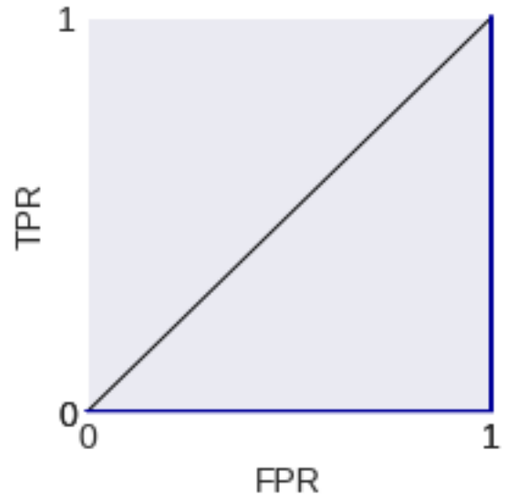



```
df['ответ'] = (df['оценка'] > 0.25).astype(int)
df.sort_values('оценка', ascending=False)
```

# ROC и AUC ROC




**наилучший (AUC=1), случайный (AUC~0.5) и наихудший (AUC=0) алгоритм**

**AUC = area under curve**

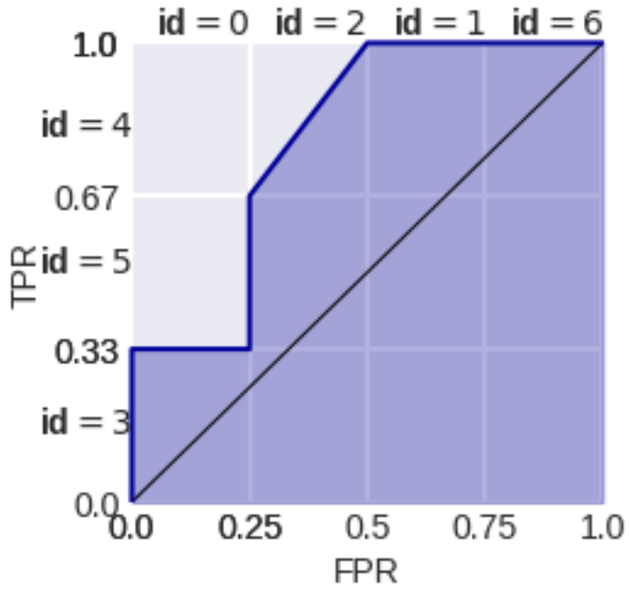
```
from sklearn.metrics import roc_curve
fpr, tpr, thresholds = roc_curve(y_test, a)
plt.plot(fpr, tpr, lw=3, c='#000099')
```

# Смысл AUC

**AUC ~ число правильно отсортированных пар**
**(на рис. «кирпичики»)**

**Это сложно объяснить заказчику!**

$$\text{AUC} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{m} I[y_i < y_j] I[a_i < a_j]}{\sum_{i=1}^{m}\sum_{j=1}^{m} I[y_i < y_j]}$$




| | оценка | класс | ответ |
|---|---|---|---|
| **3** | 0.6 | 1 | 1 |
| **0** | 0.5 | 0 | 1 |
| **5** | 0.3 | 1 | 1 |
| **2** | 0.2 | 0 | 0 |
| **4** | 0.2 | 1 | 0 |
| **1** | 0.1 | 0 | 0 |
| **6** | 0.0 | 0 | 0 |

**Чем хороша эта запись?**

**Что неправильно (требует пояснения) в формуле?**

# Смысл AUC

**Чем хороша запись?**

$$\text{AUC} = \frac{\sum_{i=1}^{m}\sum_{j=1}^{m} I[y_i < y_j]I[a_i < a_j]}{\sum_{i=1}^{m}\sum_{j=1}^{m} I[y_i < y_j]}$$

**Можно обобщить, например, на регрессию.**

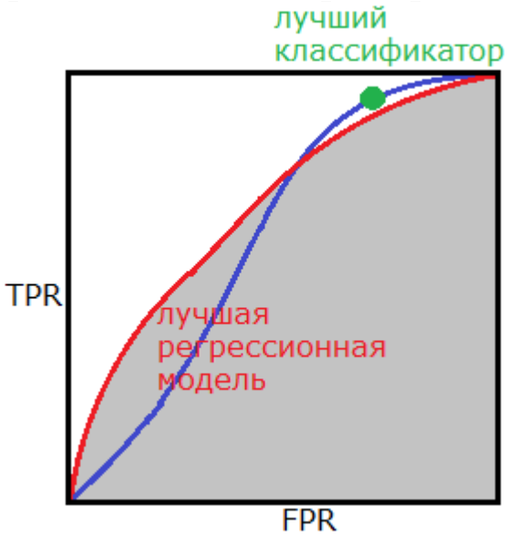
**Что неправильно (требует пояснения) в формуле?**

$$I[a_i < a_j] = \begin{cases} 1, & a_i < a_j, \\ 1/2, & a_i = a_j, \\ 0, & a_i > a_j. \end{cases}$$

# AUC ROC

+ в задачах, где важен порядок
+ учитывает разную мощность классов (не зависит от пропорций)
+ не важны значения, важен порядок
+ можно использовать для оценки признаков

– «завышает» качество
– оценивает не конкретный классификатор, а регрессию
– сложно объяснить заказчику
– не путать классификацию и регрессию

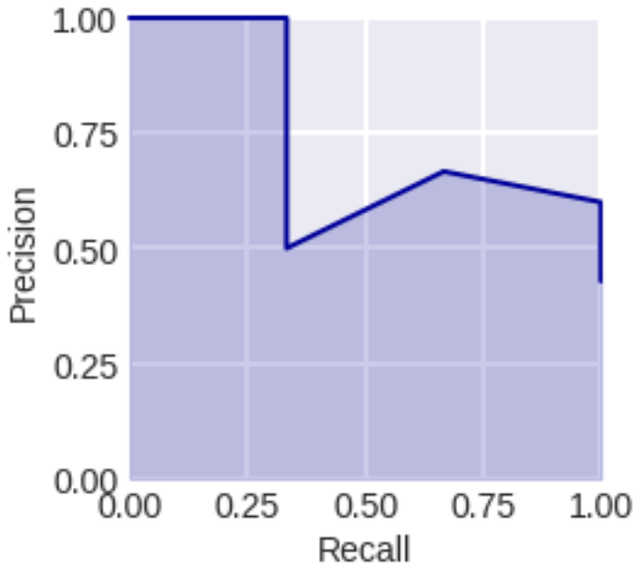


# Ещё примеры кривых… «полнота-точность»

## Площадь под кривой.. «Average Precision» (есть и другой смысл)

| | оценка | класс |
|---|---|---|
| **0** | 0.5 | 0 |
| **1** | 0.1 | 0 |
| **2** | 0.2 | 0 |
| **3** | 0.6 | 1 |
| **4** | 0.2 | 1 |
| **5** | 0.3 | 1 |
| **6** | 0.0 | 0 |

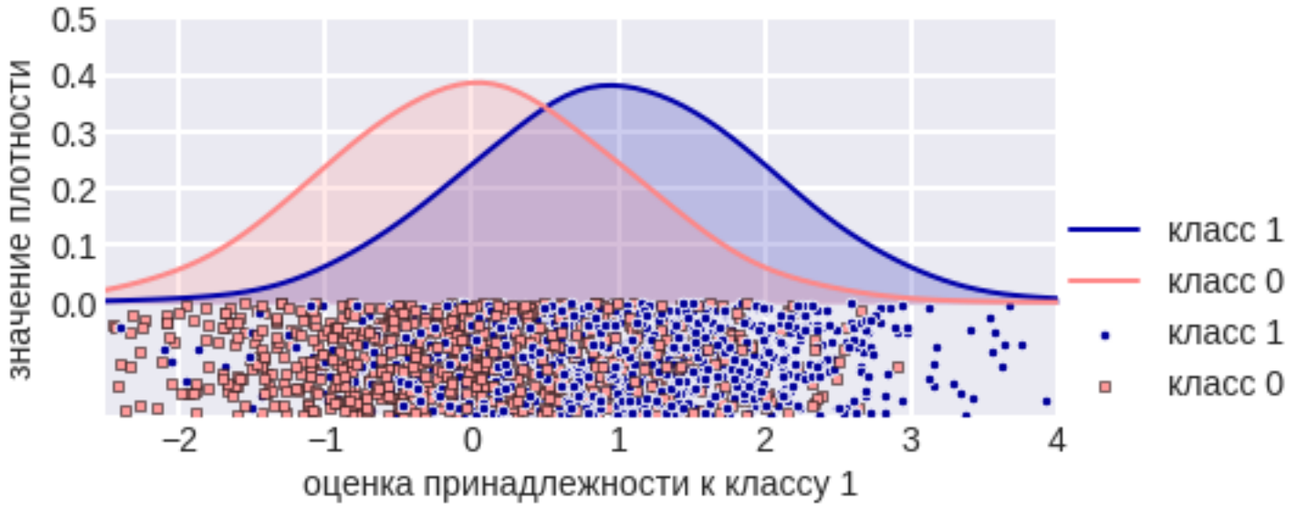
| | оценка | класс | ответ |
|---|---|---|---|
| **3** | 0.6 | 1 | 1 |
| **0** | 0.5 | 0 | 1 |
| **5** | 0.3 | 1 | 1 |
| **2** | 0.2 | 0 | 0 |
| **4** | 0.2 | 1 | 0 |
| **1** | 0.1 | 0 | 0 |
| **6** | 0.0 | 0 | 0 |

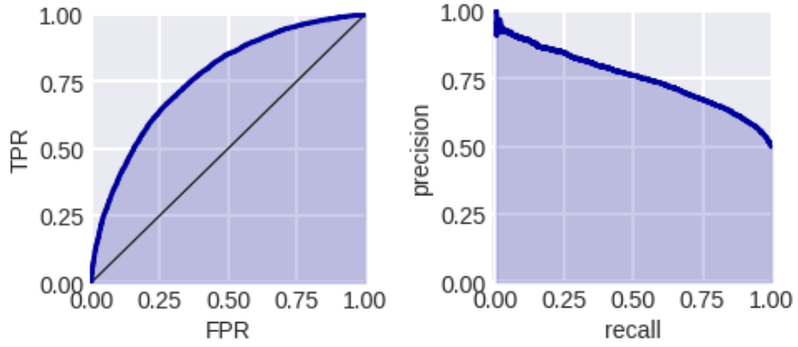



```
from sklearn.metrics import precision_recall_curve
precision, recall, thresholds = precision_recall_curve(y_test, a)
plt.plot(recall, precision)
# вычисление площади методом трапеций
from sklearn.metrics import auc
auc(recall, precision)
# или готовую функцию использовать
from sklearn.metrics import average_precision_score
```

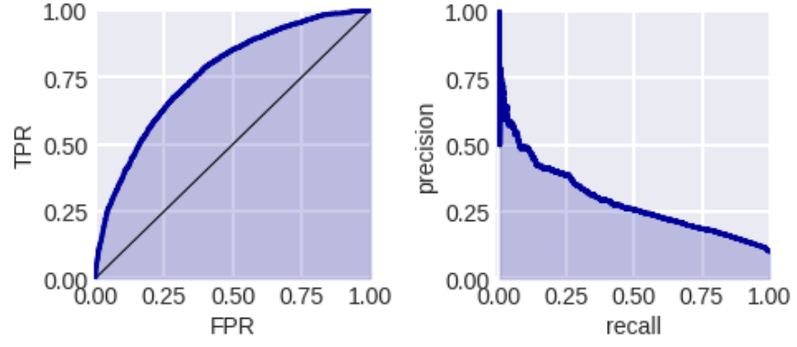
# Сравнение метрик в задачах классификации

## Модельные задачи

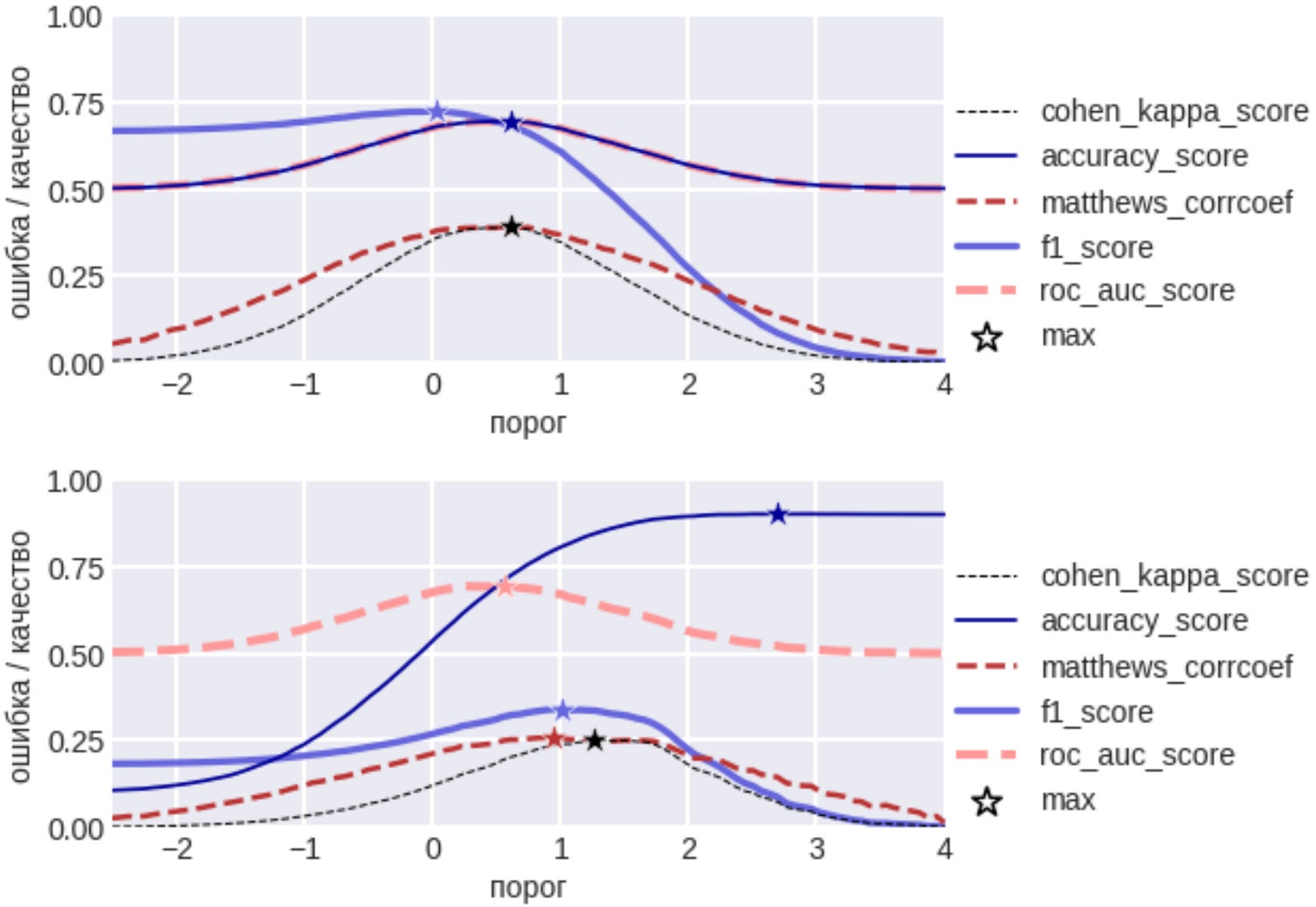






# Сравнение метрик в задачах классификации



# Итог по нечёткой бинарной классификации

**LogLoss ~ вероятность**

**ROC AUC / GINI ~ порядок**

**Есть обобщения ROC AUC**

**Есть другие кривые, например PR**


https://raw.githubusercontent.com/Dyakonov/IML/master/2020/IML2020_06scikitlearn_01.pdf

**– есть код для различных организаций контроля**