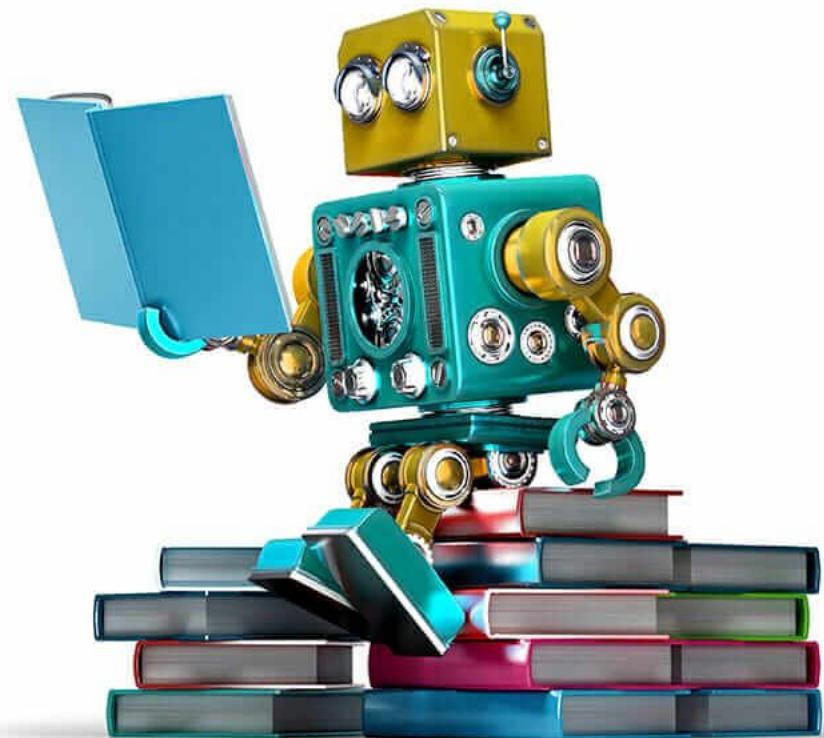


Метрики качества. Проблема переобучения.

Nadya Zueva

План

1. Оценки качества классификации
2. Критерии обобщающей способности
3. Методы отбора признаков
4. Методы работы с разреженными признаками
5. тестирование модели



Оценки качества 100
классификации 010

Confusion matrix

		Actual Value (as confirmed by experiment)	
		positives	negatives
Predicted Value (predicted by the test)	positives	TP True Positive	FP False Positive
	negatives	FN False Negative	TN True Negative

sklearn.metrics.confusion_matrix

```
sklearn.metrics. confusion_matrix(y_true, y_pred, labels=None, sample_weight=None)
```

[source]

Compute confusion matrix to evaluate the accuracy of a classification

By definition a confusion matrix C is such that $C_{i,j}$ is equal to the number of observations known to be in group i but predicted to be in group j .

Thus in binary classification, the count of true negatives is $C_{0,0}$, false negatives is $C_{1,0}$, true positives is $C_{1,1}$ and false positives is $C_{0,1}$.

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

Parameters: `y_true` : array, shape = [n_samples]

Ground truth (correct) target values.

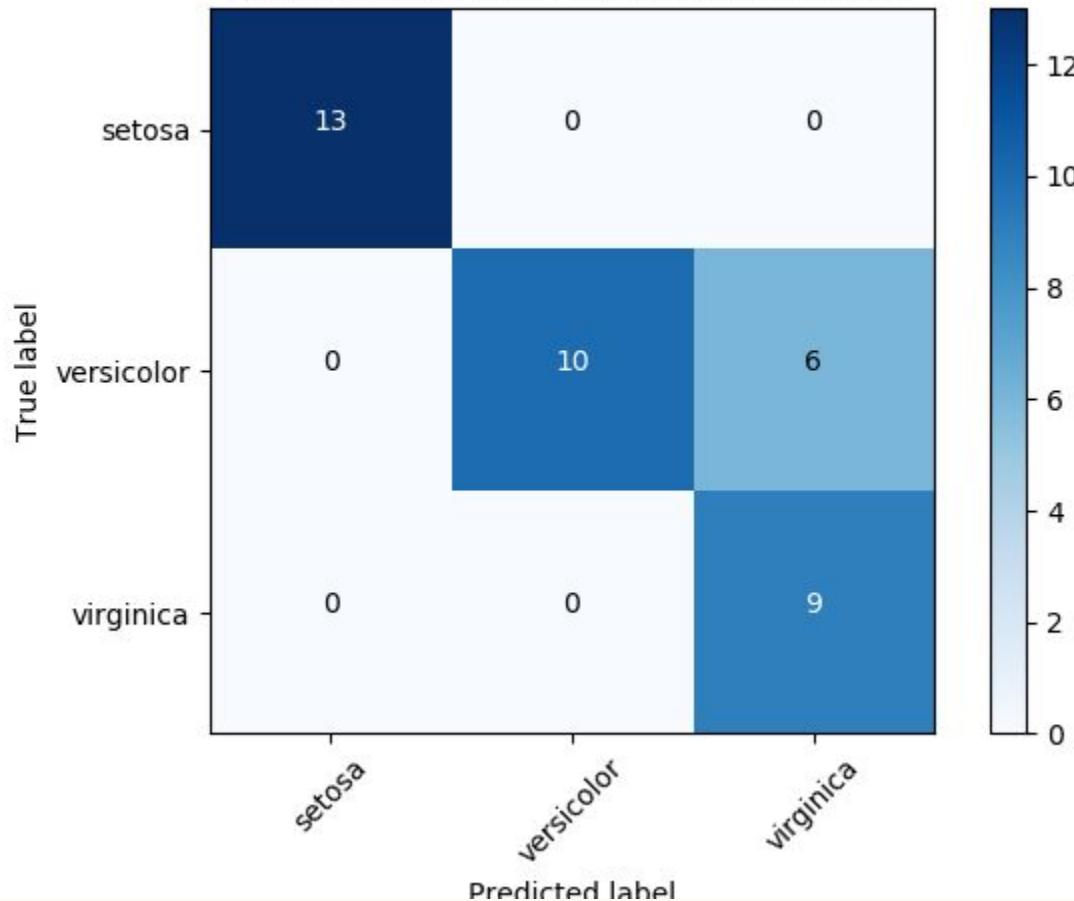
`y_pred` : array, shape = [n_samples]

Estimated targets as returned by a classifier.

`labels` : array, shape = [n_classes], optional

List of labels to index the matrix. This may be used to reorder or select a subset of labels

Confusion matrix, without normalization



Accuracy

Интуитивная, понятная и почти неиспользуемая, так как абсолютно бесполезна для работы с несбалансированными классами

$$accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$



[Previous](#) [sklearn.metrics.](#) [Next](#) [sklearn.metrics.](#) [Up](#)
[API Reference](#)

scikit-learn v0.19.2

[Other versions](#)

Please [cite us](#) if you use
the software.

[sklearn.metrics.accuracy_score](#)
Examples using
[sklearn.metrics.accuracy_score](#)

`sklearn.metrics.accuracy_score(y_true, y_pred, normalize=True, sample_weight=None)` [\[source\]](#)

Accuracy classification score.

In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in `y_true`.

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

Parameters: `y_true` : 1d array-like, or label indicator array / sparse matrix

Ground truth (correct) labels.

`y_pred` : 1d array-like, or label indicator array / sparse matrix

Predicted labels, as returned by a classifier.

`normalize` : bool, optional (default=True)

If `False`, return the number of correctly classified samples. Otherwise, return the fraction of correctly classified samples.

`sample_weight` : array-like of shape = [n_samples], optional

Sample weights.

Returns: `score` : float

[Next](#)

Пример: spam prediction

Допустим, мы хотим оценить работу спам-фильтра почты. У нас есть 100 не-спам писем, 90 из которых наш классификатор определил верно (True Negative = 90, False Positive = 10), и 10 спам-писем, 5 из которых классификатор также определил верно (True Positive = 5, False Negative = 5).

Тогда accuracy:

$$accuracy = \frac{5 + 90}{5 + 90 + 10 + 5} = 86,4$$

Однако если мы просто будем предсказывать все письма как не-спам, то получим более высокую accuracy:

$$accuracy = \frac{0 + 100}{0 + 100 + 0 + 10} = 90,9$$

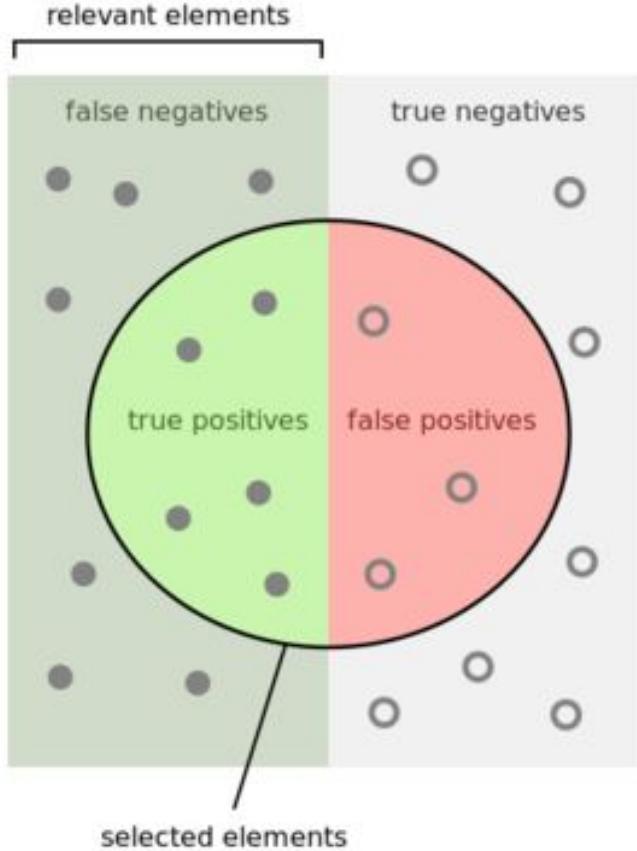
Precision, Recall, F-мера

Для оценки качества работы алгоритма на каждом из классов по отдельности введем метрики precision (точность) и recall (полнота).

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

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

Precision можно интерпретировать как долю объектов, названных классификатором положительными и при этом действительно являющимися положительными, а recall показывает, какую долю объектов положительного класса из всех объектов положительного класса нашел алгоритм.



How many selected items are relevant?

$$\text{Precision} = \frac{\text{true positives}}{\text{true positives} + \text{false positives}}$$

How many relevant items are selected?

$$\text{Recall} = \frac{\text{true positives}}{\text{true positives} + \text{false negatives}}$$



Previous
[sklearn.metrics.](#)
—
Next
[sklearn.metrics.](#)
—
Up
[API Reference](#)

scikit-learn v0.19.2
Other versions

Please [cite us](#) if you use
the software.

[sklearn.metrics.precision_score](#)
Examples using
[sklearn.metrics.precision_score](#)

«

sklearn.metrics.precision_score

`sklearn.metrics.precision_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None)`

[source]

Compute the precision

The precision is the ratio `tp / (tp + fp)` where `tp` is the number of true positives and `fp` the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.

The best value is 1 and the worst value is 0.

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

Parameters: `y_true` : 1d array-like, or label indicator array / sparse matrix

Ground truth (correct) target values.

`y_pred` : 1d array-like, or label indicator array / sparse matrix

Estimated targets as returned by a classifier.

`labels` : list, optional

The set of labels to include when `average != 'binary'`, and their order if `average` is `None`. Labels present in the data can be excluded, for example to calculate a multiclass average ignoring a majority negative class; while labels not present in the data will result

Next



[Previous
sklearn.metrics](#) [Next
sklearn.metrics](#) [Up
API Reference](#)

scikit-learn v0.19.2

[Other versions](#)

Please [cite us](#) if you use
the software.

[sklearn.metrics.recall_score](#)
Examples using
[sklearn.metrics.recall_score](#)

«

sklearn.metrics.recall_score

`sklearn.metrics. recall_score(y_true, y_pred, labels=None, pos_label=1, average='binary', sample_weight=None)`

[\[source\]](#)

Compute the recall

The recall is the ratio `tp / (tp + fn)` where `tp` is the number of true positives and `fn` the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.

The best value is 1 and the worst value is 0.

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

Parameters: `y_true` : 1d array-like, or label indicator array / sparse matrix

Ground truth (correct) target values.

`y_pred` : 1d array-like, or label indicator array / sparse matrix

Estimated targets as returned by a classifier.

`labels` : list, optional

The set of labels to include when `average != 'binary'`, and their order if `average is None`. Labels present in the data can be excluded, for example to calculate a multiclass average ignoring a majority negative class. While labels not present in the data will result

[Next](#)

F-мера

Существует несколько различных способов объединить precision и recall в агрегированный критерий качества. F-мера (в общем случае F_β) — среднее гармоническое precision и recall :

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

[Previous
sklearn.mod...
el...](#)[Next
sklearn.mod...
el...](#)[Up
API
Reference](#)[scikit-learn v0.19.2](#)[Other versions](#)

Please [cite us](#) if you use
the software.

[sklearn.model_selection.GridSearchCV](#)
Examples using
[sklearn.model_selection.GridS...](#)

```
class sklearn.model_selection.GridSearchCV(estimator, param_grid, scoring=None, fit_params=None,  
n_jobs=1, iid=True, refit=True, cv=None, verbose=0, pre_dispatch='2*n_jobs', error_score='raise',  
return_train_score='warn')
```

[\[source\]](#)

Exhaustive search over specified parameter values for an estimator.

Important members are `fit`, `predict`.

`GridSearchCV` implements a “fit” and a “score” method. It also implements “predict”, “predict_proba”, “decision_function”, “transform” and “inverse_transform” if they are implemented in the estimator used.

The parameters of the estimator used to apply these methods are optimized by cross-validated grid-search over a parameter grid.

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

Parameters: `estimator` : estimator object.

This is assumed to implement the scikit-learn estimator interface. Either estimator needs to provide a `score` function, or `scoring` must be passed.

`param_grid` : dict or list of dictionaries

Dictionary with parameters names (string) as keys and lists of parameter settings to try as values, or a list of such dictionaries, in which case the grids spanned by each dictionary in the list are explored. This enables searching over any sequence of parameter settings.

[Next](#)

The screenshot shows the homepage of the imbalanced-learn documentation. The left sidebar is dark blue with white text, listing navigation links. The main content area is white with black text and features several sections with blue headings.

imbalanced-learn
0.3.0

Search docs

GETTING STARTED

- Install and contribution

DOCUMENTATION

- User Guide
- imbalanced-learn API

TUTORIAL - EXAMPLES

- General examples
- Examples based on real world datasets
- Dataset examples
- Evaluation examples
- Model Selection

ADDITIONAL INFORMATION

- Release history
- About us

Docs » Welcome to imbalanced-learn documentation!

[View page source](#)

Welcome to imbalanced-learn documentation!

Getting started

Information to install, test, and contribute to the package.

User Guide

The main documentation. This contains an in-depth description of all algorithms and how to apply them.

API Documentation

The exact API of all functions and classes, as given in the docting. The API documents expected types and allowed features for all functions, and all parameters available for the algorithms.

Examples

A set of examples illustrating the use of the different algorithms. It complements the [User Guide](#).

What's new

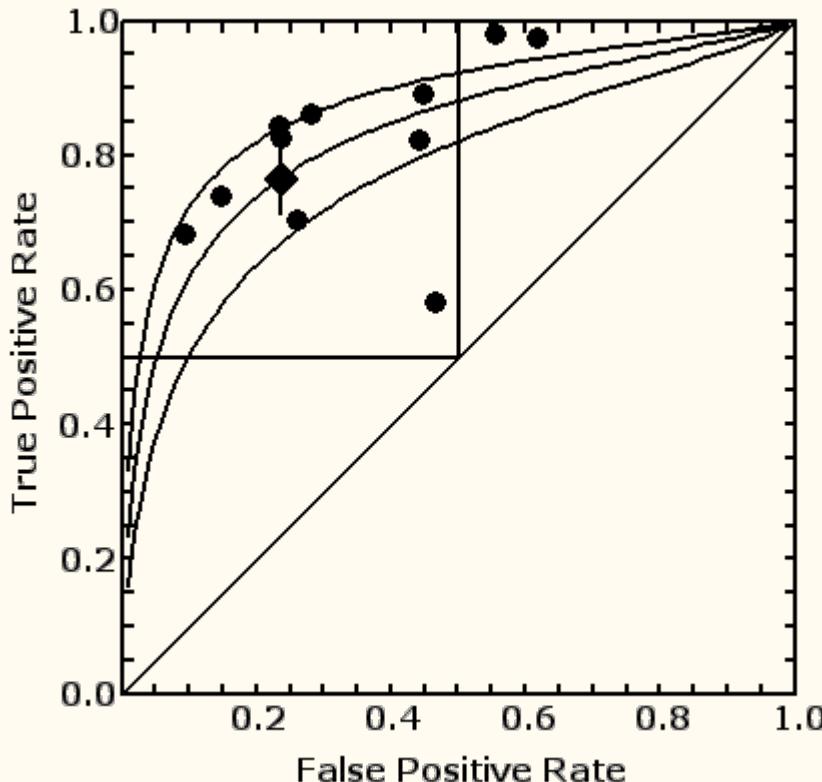
ROC-AUC

При конвертации вещественного ответа алгоритма в бинарную метку, мы должны выбрать какой-либо порог, при котором 0 становится 1. Естественным и близким кажется порог, равный 0.5, но он не всегда оказывается оптимальным, например, при отсутствии баланса классов.

Одним из способов оценить модель в целом, не привязываясь к конкретному порогу, является AUC-ROC — площадь (Area Under Curve) под кривой ошибок (Receiver Operating Characteristic curve).

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

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



Кривая ошибок ROC (receiver operating characteristic).

Каждая точка кривой соответствует некоторому $a(x; w, w_0)$.

- по оси X: *доля ошибочных положительных классификаций* (FPR — false positive rate):

$$\text{FPR}(a, X^\ell) = \frac{\sum_{i=1}^{\ell} [y_i = -1] [a(x_i; w, w_0) = +1]}{\sum_{i=1}^{\ell} [y_i = -1]},$$

$1 - \text{FPR}(a)$ называется *специфичностью* алгоритма a .

- по оси Y: *доля правильных положительных классификаций* (TPR — true positive rate):

$$\text{TPR}(a, X^\ell) = \frac{\sum_{i=1}^{\ell} [y_i = +1] [a(x_i; w, w_0) = +1]}{\sum_{i=1}^{\ell} [y_i = +1]},$$

$\text{TPR}(a)$ называется также *чувствительностью* алгоритма a .

```
iris = datasets.load_iris()
X = iris.data
y = iris.target

# Binarize the output
y = label_binarize(y, classes=[0, 1, 2])
n_classes = y.shape[1]

# Add noisy features to make the problem harder
random_state = np.random.RandomState(0)
n_samples, n_features = X.shape
X = np.c_[X, random_state.randn(n_samples, 200 * n_features)]

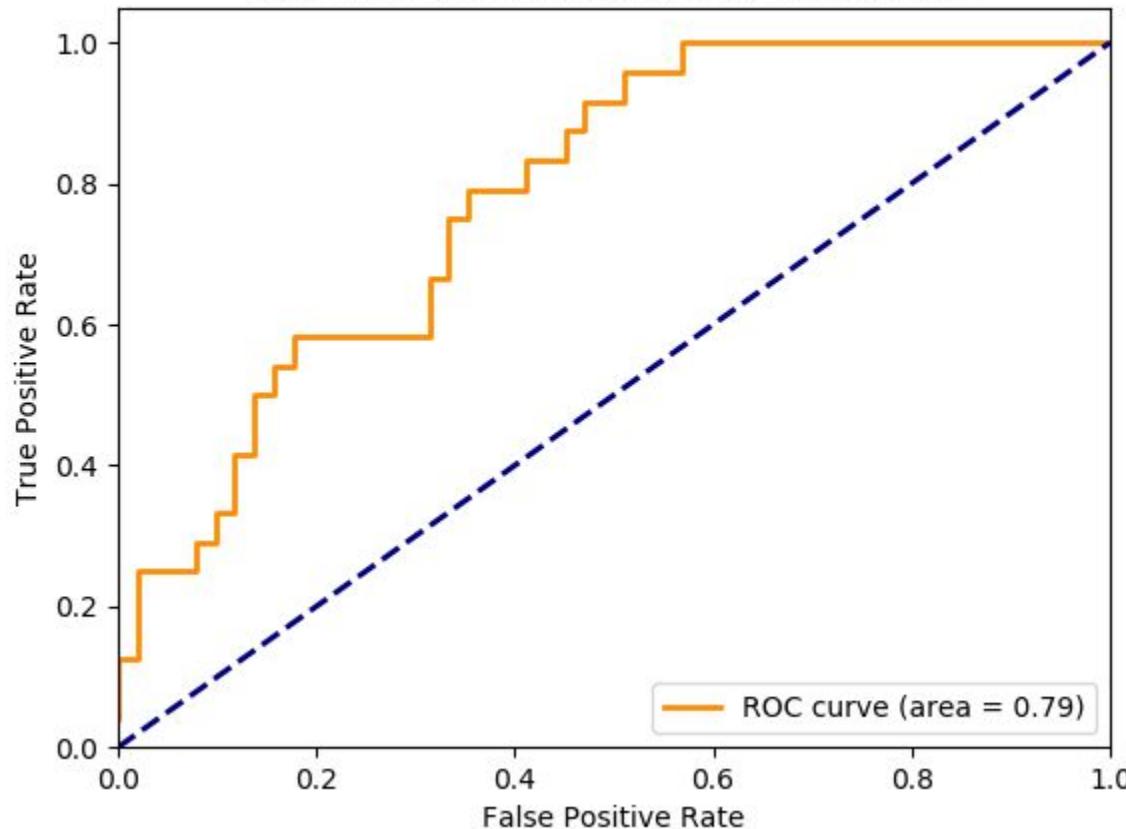
# shuffle and split training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.5,
                                                    random_state=0)

# Learn to predict each class against the other
classifier = OneVsRestClassifier(svm.SVC(kernel='linear', probability=True,
                                           random_state=random_state))
y_score = classifier.fit(X_train, y_train).decision_function(X_test)

# Compute ROC curve and ROC area for each class
fpr = dict()
tpr = dict()
roc_auc = dict()
for i in range(n_classes):
    fpr[i], tpr[i], _ = roc_curve(y_test[:, i], y_score[:, i])
    roc_auc[i] = auc(fpr[i], tpr[i])

# Compute micro-average ROC curve and ROC area
fpr["micro"], tpr["micro"], _ = roc_curve(y_test.ravel(), y_score.ravel())
roc_auc["micro"] = auc(fpr["micro"], tpr["micro"])
```

Receiver operating characteristic example

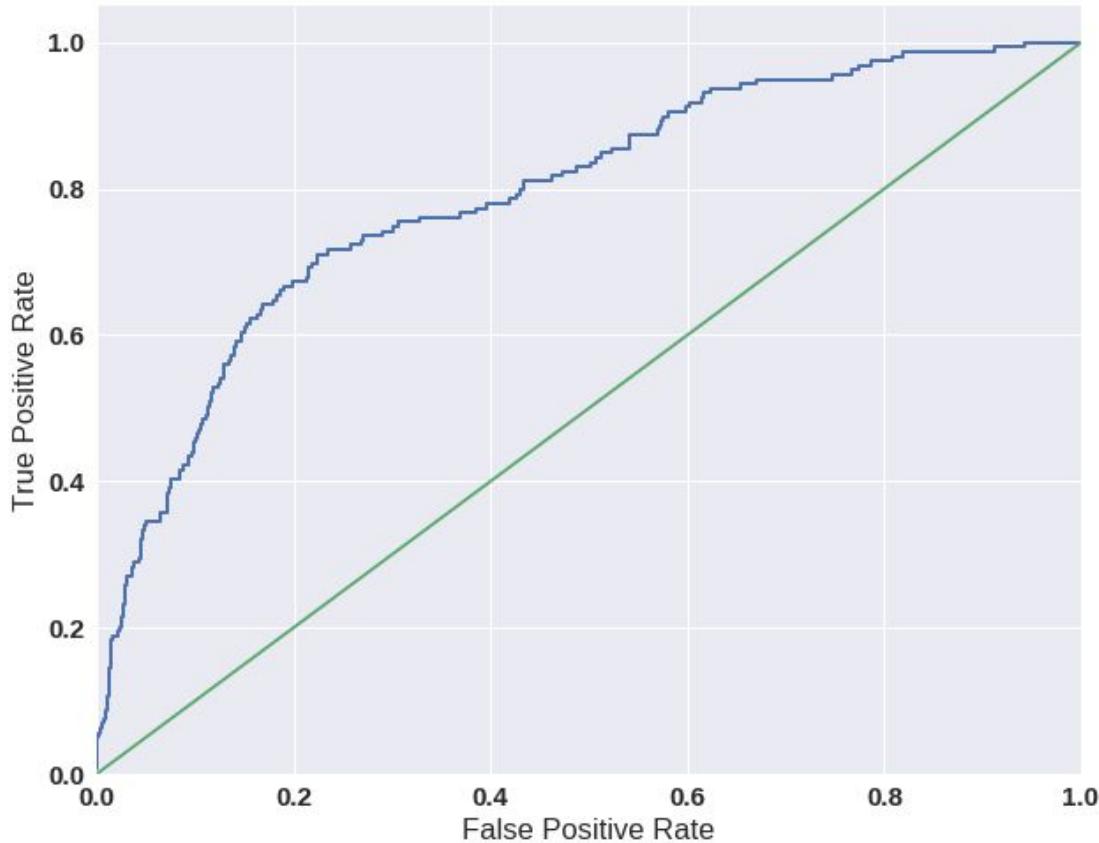


ROC-AUC

AUC -- площадь под графиком ROC.

Критерий AUC-ROC устойчив к несбалансированным классам и может быть интерпретирован как вероятность того, что случайно выбранный **positive** объект будет отранжирован классификатором выше (будет иметь более высокую вероятность быть **positive**), чем случайно выбранный **negative** объект.

ROC curve



- **Алгоритм 1** возвращает 100 документов, 90 из которых релевантны. Таким образом,

$$TPR = \frac{TP}{TP + FN} = \frac{90}{90 + 10} = 0.9$$

$$FPR = \frac{FP}{FP + TN} = \frac{10}{10 + 999890} = 0.00001$$

- **Алгоритм 2** возвращает 2000 документов, 90 из которых релевантны. Таким образом,

$$TPR = \frac{TP}{TP + FN} = \frac{90}{90 + 10} = 0.9$$

$$FPR = \frac{FP}{FP + TN} = \frac{1910}{1910 + 997990} = 0.00191$$

Разница в False Positive Rate между этими двумя алгоритмами *крайне* мала — всего **0.0019**.

Это является следствием того, что AUC-ROC измеряет долю False Positive относительно True Negative и в задачах, где нам не так важен больший класс, может давать не совсем адекватную картину при сравнении алгоритмов.

- Алгоритм 1

$$precision = \frac{TP}{TP + FP} = 90 / (90 + 10) = 0.9$$

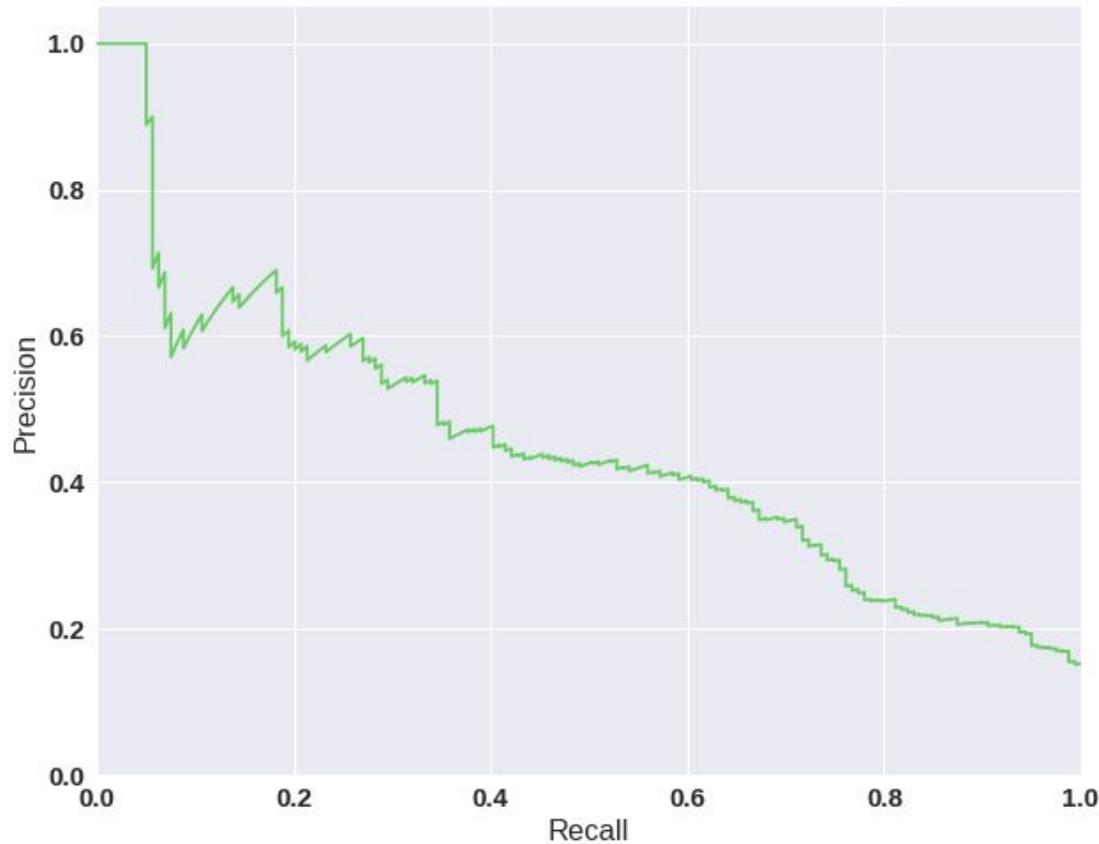
$$recall = \frac{TP}{TP + FN} = 90 / (90 + 10) = 0.9$$

- Алгоритм 2

$$precision = \frac{TP}{TP + FP} = \frac{90}{90 + 1910} = 0.045$$

$$recall = \frac{TP}{TP + FN} = \frac{90}{90 + 10} = 0.9$$

Precision-Recall curve



Выводы

- В случае многоклассовой классификации нужно внимательно следить за метриками каждого из классов и следовать логике решения задачи, а не оптимизации метрики
- В случае неравных классов нужно подбирать баланс классов для обучения и метрику, которая будет корректно отражать качество классификации
- Выбор метрики нужно делать с фокусом на предметную область, предварительно обрабатывая данные и, возможно, сегментируя (как в случае с делением на богатых и бедных клиентов)

Выбор метода обучения

100
010

Дано: X — пространство объектов; Y — множество ответов;
 $X^\ell = (x_i, y_i)_{i=1}^\ell$ — обучающая выборка, $y_i = y^*(x_i)$;
 $A_t = \{a: X \rightarrow Y\}$ — модели алгоритмов, $t \in T$;
 $\mu_t: (X \times Y)^\ell \rightarrow A_t$ — методы обучения, $t \in T$.

Найти: метод μ_t с наилучшей обобщающей способностью.

Частные случаи:

- выбор лучшей модели A_t (model selection);
- выбор метода обучения μ_t для заданной модели A (в частности, оптимизация гиперпараметров);
- отбор признаков (features selection):
 $F = \{f_j: X \rightarrow D_j: j = 1, \dots, n\}$ — множество признаков;
метод обучения μ_J использует только признаки $J \subseteq F$.

Как оценить качество обучения?

$\mathcal{L}(a, x)$ — функция потерь алгоритма a на объекте x ;

$Q(a, X^\ell) = \frac{1}{\ell} \sum_{i=1}^{\ell} \mathcal{L}(a, x_i)$ — функционал качества a на X^ℓ .

Внутренний критерий оценивает качество на обучении X^ℓ :

$$Q_\mu(X^\ell) = Q(\mu(X^\ell), X^\ell).$$

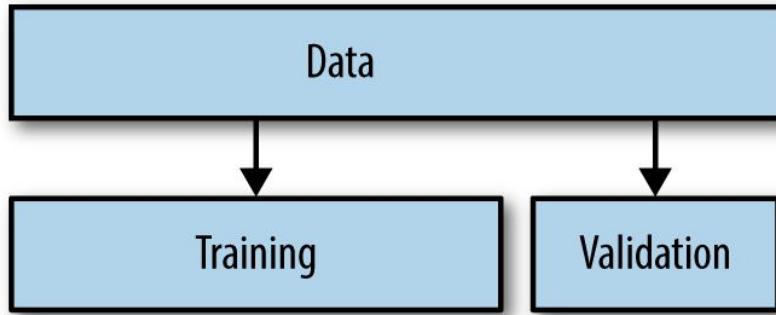
Недостаток: эта оценка смещена, т.к. μ минимизирует её же.

Внешний критерий оценивает качество «вне обучения»,
например, по отложенной (hold-out) контрольной выборке X^k :

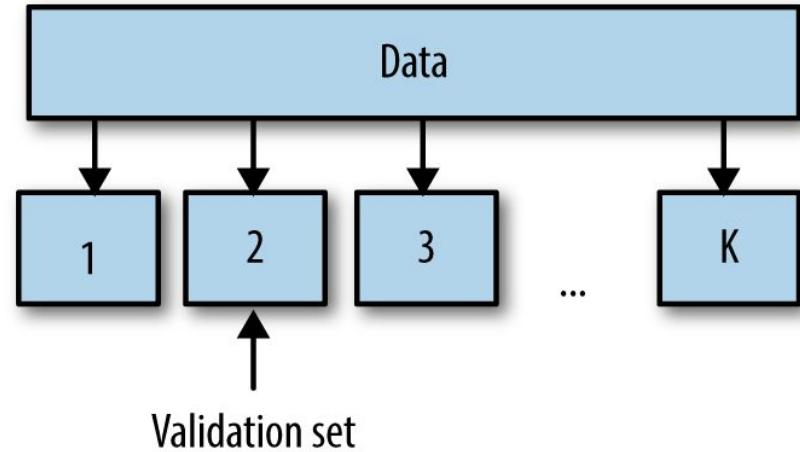
$$Q_\mu(X^\ell, X^k) = Q(\mu(X^\ell), X^k).$$

Недостаток: эта оценка зависит от разбиения $X^L = X^\ell \sqcup X^k$.

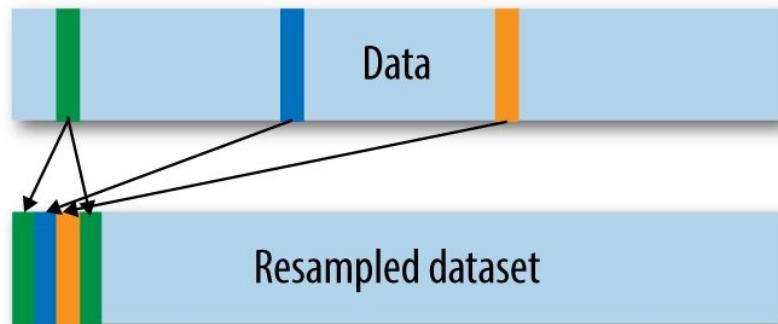
Hold-out validation



K-fold cross validation

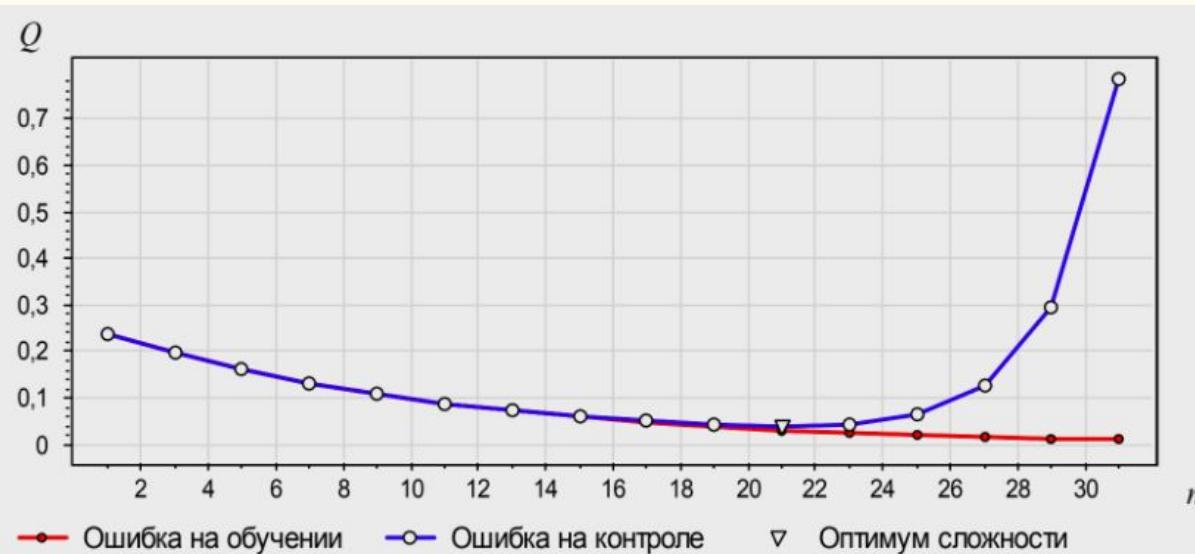


Bootstrap resampling



Внутренний VS Внешний критерий

Ошибка на внутреннем монотонно убывает, в то время как на внешнем имеется характерный минимум



Cross-Validation

Усреднение оценок hold-out по заданному N — множеству разбиений $X^L = X_n^\ell \sqcup X_n^k$, $n = 1, \dots, N$:

$$CV(\mu, X^L) = \frac{1}{|N|} \sum_{n \in N} Q_\mu(X_n^\ell, X_n^k).$$

Способы задания N :

1. Случайное множество разбиений
2. Полная проверка (слишком сложно, используются комбинаторные оценки)

- 3.

$$LOO(\mu, X^L) = \frac{1}{L} \sum_{i=1}^L Q_\mu(X^L \setminus \{x_i\}, \{x_i\}).$$

leave-one-out

- 4.

$$CV_q(\mu, X^L) = \frac{1}{q} \sum_{n=1}^q Q_\mu(X^L \setminus X_n^{\ell_n}, X_n^{\ell_n}).$$

Q-fold CV

q -fold t -times

$$CV_{t \times q}(\mu, X^L) = \frac{1}{t} \sum_{s=1}^t \frac{1}{q} \sum_{n=1}^q Q_\mu(X^L \setminus X_{sn}^{\ell_n}, X_{sn}^{\ell_n}).$$

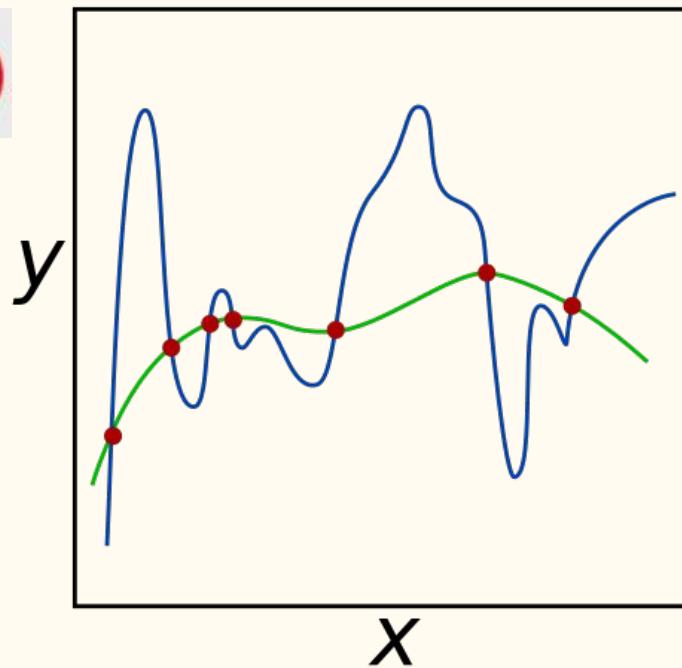
Преимущества $t \times q$ -fold CV:

- увеличением t можно улучшать точность оценки (компромисс между точностью и временем вычислений);
- каждый объект участвует в контроле ровно t раз;
- оценивание доверительных интервалов (95% при $t = 40$).

Критерий регуляризации

Регуляризатор -- аддитивная добавка к критерию за сложность модели A

$$Q_{\text{рег}}(\mu, X^\ell) = Q_\mu(X^\ell) + \text{штраф}(A)$$



L_2 -регуляризация (ридж-регрессия):

$$\text{штраф}(w) = \tau \|w\|_2^2 = \tau \sum_{j=1}^n w_j^2.$$

Ridge-регрессия

```
>>> from sklearn.linear_model import Ridge
>>> import numpy as np
>>> n_samples, n_features = 10, 5
>>> np.random.seed(0)
>>> y = np.random.randn(n_samples)
>>> X = np.random.randn(n_samples, n_features)
>>> clf = Ridge(alpha=1.0)
>>> clf.fit(X, y)
Ridge(alpha=1.0, copy_X=True, fit_intercept=True, max_iter=None,
      normalize=False, random_state=None, solver='auto', tol=0.001)
```

Methods

<code>fit (X, y[, sample_weight])</code>	Fit Ridge regression model
<code>get_params ([deep])</code>	Get parameters for this estimator.
<code>predict (X)</code>	Predict using the linear model
<code>score (X, y[, sample_weight])</code>	Returns the coefficient of determination R^2 of the prediction.
<code>set_params (**params)</code>	Set the parameters of this estimator.

L_1 -регуляризация (LASSO):

$$\text{штраф}(w) = \tau \|w\|_1 = \tau \sum_{j=1}^n |w_j|.$$

Lasso-регуляризация

```
>>> from sklearn import linear_model
>>> clf = linear_model.Lasso(alpha=0.1)
>>> clf.fit([[0,0], [1, 1], [2, 2]], [0, 1, 2])
Lasso(alpha=0.1, copy_X=True, fit_intercept=True, max_iter=1000,
      normalize=False, positive=False, precompute=False, random_state=None,
      selection='cyclic', tol=0.0001, warm_start=False)
>>> print(clf.coef_)
[ 0.85  0. ]
>>> print(clf.intercept_)
0.15
```

Methods

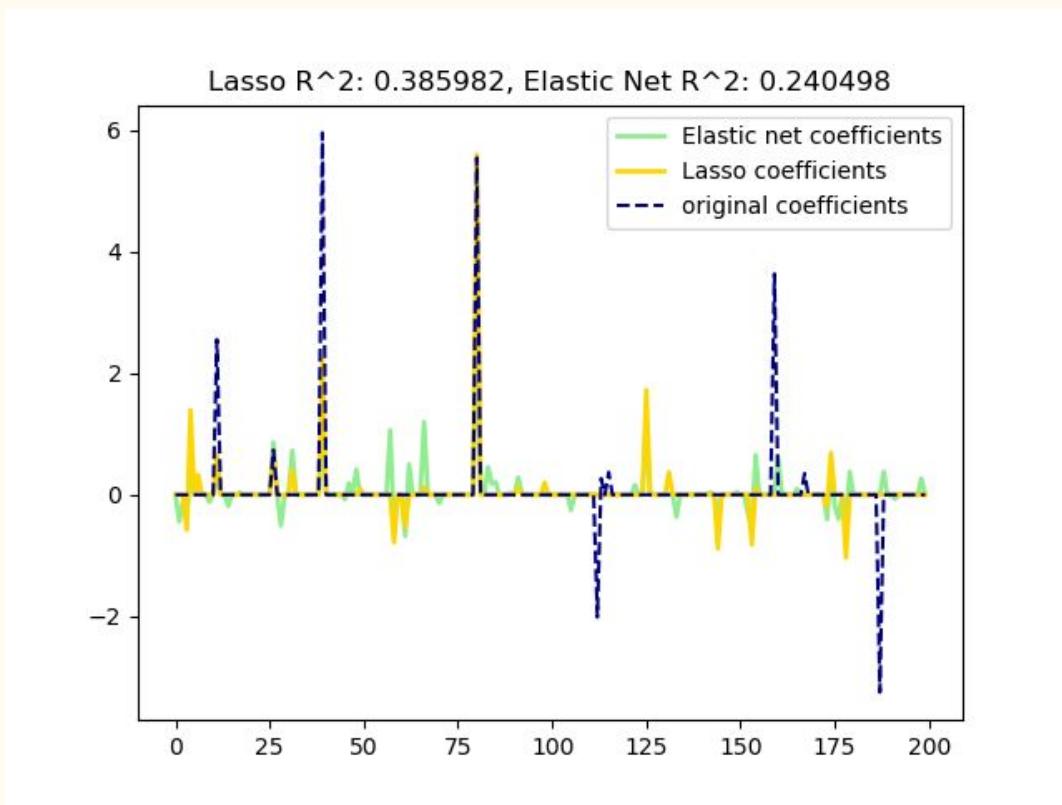
<code>fit (X, y[, check_input])</code>	Fit model with coordinate descent.
<code>get_params ([deep])</code>	Get parameters for this estimator.
<code>path (X, y[, l1_ratio, eps, n_alphas, ...])</code>	Compute elastic net path with coordinate descent
<code>predict (X)</code>	Predict using the linear model
<code>score (X, y[, sample_weight])</code>	Returns the coefficient of determination R^2 of the prediction.
<code>set_params (**params)</code>	Set the parameters of this estimator.

`__init__ (alpha=1.0, fit_intercept=True, normalize=False, precompute=False, copy_X=True, max_iter=1000, tol=0.0001, warm_start=False, positive=False, random_state=None, selection='cyclic')` [\[source\]](#)

`fit (X, y, check_input=True)`

[\[source\]](#)

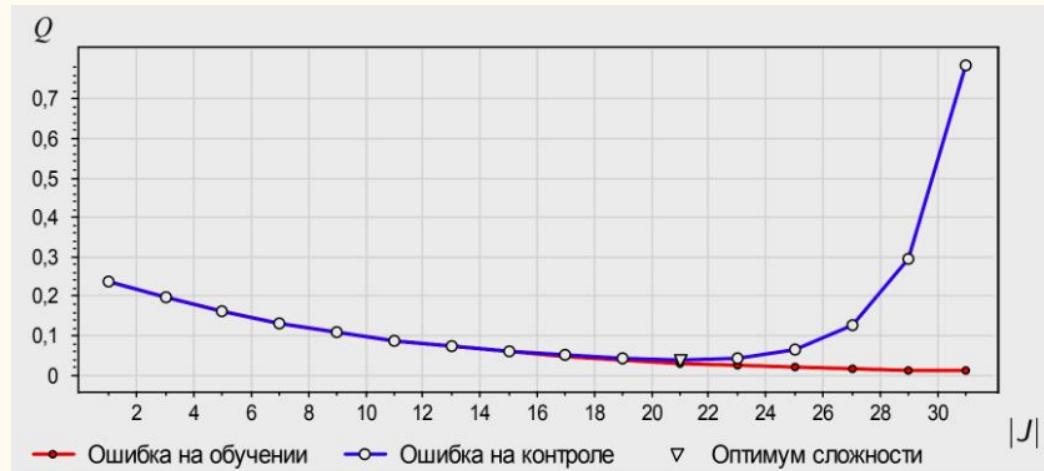
Elastic Net



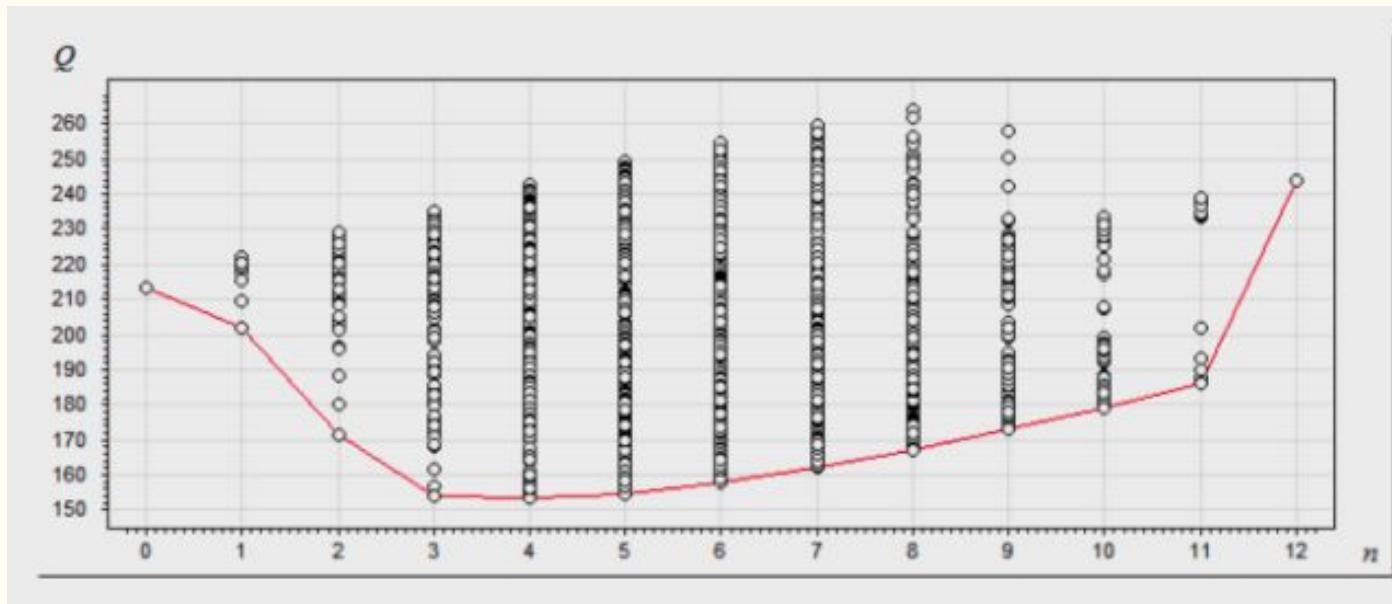
Задача отбора признаков

F -- набор признаков, M_j -- метод обучения, использующий только j выделенных признаков из F . X -- обучающая выборка.

$$Q(j) = Q(M_j, X) \rightarrow \min$$



Полный перебор



Эвристики

1. Жадный алгоритм отбора признаков
2. Чередование добавлений и удалений признаков
3. BFS
4. Эволюционный алгоритм поиска
 - a. Увеличивать вероятность перехода от более успешного родителя
 - b. Применение совокупности критериев
 - c. При стагнации -- мутации
 - d. Параллельно выращивать несколько изолированных популяций

